The 26th International Conference on Automated Planning and Scheduling

June 12-17, 2016. London, UK

Doctoral Consortium Dissertation Abstracts

The Doctoral Consortium brings together junior and experienced researchers in planning, scheduling and related areas from across the globe, in particular those studying for a Doctoral degree. It provides a forum for networking with the ICAPS community in an informal, social setting. The Doctoral Consortium will be held as a full day workshop on June 12th, 2016.
The program includes an invited talk on research skills and career development, the opportunity for participant to give a short presentation, and a poster session during the main conference.

DC Chairs:

- Peter Gregory (University of Teeside, UK)
- Lee McCluskey (University of Huddersfield, UK)
- Fabio Mercorio (University of Milan-Bicocca, Italy)

Abstracts

1 Heuristic Search and Applications 3

Automated Planning and Scheduling E0 Constellations' Operations using Ant Colony Optimization . . . 4
Evridiki Ntagiou

Solver Paramter Tuning and Runtime Predictions of Flexible Hybrid Mathematical models 10
Michael Barry

Constructing Heuristics for PDDL+ Planning Domains 15
Wiktor Piotrowski

Risk-Sensitive Planning with Dynamic Uncertainty 21
Liana Marinescu

2 Multi Agent Planning & Plan Execution 27

A Distributed Online Multi-Agent Planning System 28
Rafael Cardoso

Integrating Planning and Recognition to Close the Interaction Loop 34
Rick Freedman

Distributed Privacy-preserving Multi-agent Planning 39
Andrea Bonisoli

Planning with Concurrent Execution 44
Bence Cserna

3 Temporal Planning 47

Mixed Discrete-Continuous Planning with Complex Behaviors 48
Enrique Fernandez Golzalez

Planning with Flexible Timelines in the Real World 54
Alessandro Umbrico

POPCorn: Planning with Constrained Real Numerics 60
Emre Savas

Planning with PDDL3.0 Preferences by Compilation into STRIPS with Action Costs 66
Francesco Percassi

Planning Under Uncertainty with Temporally Extended Goals 71
Alberto Camacho

Temporal Inference In Forward Search Temporal Planning 73
Atif Talukdar

4 Planning and Scheduling 79

Task Scheduling and Trajectory Generation of Multiple Intelligent Vehicles **80**
Jennifer David

Decoupled State Space Search **83**
Daniel Gnad

Hierarchical Task Model with Alternatives for Predictive-reactive Scheduling **89**
Marek Vlk

Numeric Planning **94**
Johannes Aldinger

Exploiting Search Space Structure in Classical Planning: Analyses and Algorithms **97**
Matasaru Asai

SAT/SMT techniques for planning problems **102**
Joan Espasa Arxer

5 Planning under Uncertainty and Applications **106**

Robotic control through model-free reinforcement learning **107**
Ludovic Hofer

Exploiting Symmetries in Sequential Decision Making under Uncertainty **109**
Ankit Anand

Recommending and Planning Trip Itineraries for Individual Travellers and Groups of Tourists **115**
Kwan Hui LIM

Constructing Plan Trees for Simulated Penetration Testing **121**
Dorin Shmaryahu

Optimization Approaches to Multi-robot Planning and Scheduling **128**
Kyle Booth

6 Knowledge Engineering and Applications **131**

Learning Static Constraints for Domain Modeling from Training Plans **132**
Rabia Jilani

Using GORE method for Requirement Engineering of Planning & Scheduling **137**
Javier Martinez

Critical Constrained Planning and an Application to Network Penetration Testing **141**
Marcel Steinmetz

Human-Robot Communication in Automated Planning **145**
Aleck MacNally

Session 1

Heuristic Search and Applications

Automated Planning and Scheduling EO Constellations' Operations with Ant Colony Optimization

Evridiki Vasileia Ntagiou

Surrey Space Centre, University of Surrey

Abstract

In this work we are interested in Automating the process of Planning and Scheduling the operations of an Earth Observation constellation. To this respect, we represent the problem with a directed graph and use Ant Colony Optimization technique to find the optimal solution. In order to verify the quality of the solution, we employ a dynamical system. We check the scalability of the software system performing simulations. We discuss the next steps of this work which involve the coordination of multiple spacecraft by means of stigmergy and the consideration of more than one objectives that need to be optimized.

Motivation and Scope

The increasing interest in the design and development of space missions consisting of multiple coordinated spacecraft cannot be missed, in recent years. Ranging from low cost due to less system reliability requirements, to giving man the ability to perform concurrent scientific observations, the advantages of using constellations of spacecraft have attracted the complete attention of the Space community [T. A. Wagner et al.]. The Earth Observation market, in specific, is expected to grow at a rate of 16% per year over the next decade [N. Muscettola et al.]. The current trend is towards constellations consisting of many small satellites, with an increasing number of start-up companies aiming at launching such constellations of 20 hundreds or more mini-satellites. [G. Richardson et al.][E. Buchen]

The reduction of the satellites' size and corresponding shrinking of their cost has allowed many end users to benefit from data coming from satellites. Since we are dealing with the cooperation of numerous miniaturized satellites of simple capabilities, which altogether form a very complex system, the need to automate its management arises. Traditional techniques have failed to cope with such a level of complexity. *Planning and scheduling* (P&S) the operations of an EO satellite is the process of determining the time when the satellite performs specific arranged tasks, as the available resources, images' collection goals, weather condition and user requirements evolve. More specifically, the P& S system is responsible for coordinating a constellation's satellites' activities in order for the total value of the downlinked data to be maximized. The Earth observing satellites (EOSs) picture the Earth's surface, in order to satisfy an assigned goal, which in our case will be the imaging of the *Area of Interest* (AoI). EOSs can acquire images, while moving on their usually low altitude orbits. The acquired data will then need to be transmitted to the ground station. Until that is possible, the data are stored in the limited on-board memory of the satellites, limiting the images that can be acquired before the downlink.

There is a wide interest for automating the P&S process in the EO field, emanating not only from research organizations and universities [C. Iacopino et al], but also from commercial operators and agencies [S. A. Chien et al.]. The main benefit of autonomy in the planning & scheduling field is in being able to gain maximum value from the spacecraft by maximizing the use of on board resources and providing a greater level of responsiveness to sudden changes of priority, such as when natural disasters strike. Automating the P&S process of an Earth Observation mission involves *optimization* and *coordination*. It is a combinatorial optimization problem that takes place in an uncertain dynamic environment. The development of an automated P & S system also follows the needs of the upcoming missions. These employ dozens of agile satellites, where a change of attitude translates to a tilt of the imager. We consider agile EOSs that can be steered up to 45° off-nadir in the roll axis.

An EO mission may have a single goal e.g. maximize the imaged area, and many constraints, e.g. resource or weather constraints. It could also have multiple goals which are conflicting e.g. maximize the imaged area, while minimizing the resource used, and again numerous constraints. In fact, the nature of the problem is such that it includes many constraints, when realistic scenarios are studied. In most of the studies, a single-objective optimization problem with numerous constraints is considered. This alone, means that our solution will be valid under several assumptions. In order to lift those assumptions we try to decrease the number of constraints and increase the number of goals. In this case, the P&S problem is a multi-objective optimization problem. In order for a mission to be successful, the trade-off among the several objectives needs to be studied and a solution depending on the user requirements needs to be produced.

The main challenges that arise when developing a software system that is meant to be autonomous can be grouped in three main categories: Reliability, Scalability and Adaptability. When dealing with a continuously changing environment like space, a system must be able to quickly adapt to new circumstances and adjust its output correspondingly, not allowing the changes to interfere with the quality of the solution. The case is the same, when the users' preferences or the platform's characteristics change, e.g. increase in the dimensions of the Area of Interest for an EO satellite, or increase in the number of satellites available for a task in a constellation. In order to address these challenges we propose a *self-organising* architecture for the software system and a *dynamical system*, by which we can analytically study the optimization method and guarantee the convergence to a solution. Furthermore, the method we employ can easily be extended to a study of the multi-objective nature of the problem.

Novelty

This Ph.D. seeks to make contributions in three areas:

1. Development of a Self – organizing software tool as an Automated Mission Planning System. In our novel approach, we will employ a multi-agent system to manage the coordination among the constellation's spacecraft.
2. Modelling of a Probabilistic optimization technique with a nonlinear dynamical system. We formally verify the reliability of our algorithm employing a non-linear dynamical system to model its behaviour.
3. Multi objective Optimization techniques using Swarm Intelligence methods. Ant Colony Optimization technique has not been widely employed for multiple objectives optimization purposes.

Problem definition

We consider a large area of polygon shape in the surface of the Earth that we are interested in imaging, within a specific time window. What is the best way to cover the Area of Interest (AoI) with the satellites' swaths? This is a *coverage planning* problem. *Optimally* planning the images' acquisitions and assigning them to the satellites of a constellation is a combinatorial optimization problem. In order to quantify the level of optimality, we need to introduce an objective function.

This problem is of highly dynamic nature. This is due to the constant changes regarding the user requests, the weather conditions, e.t.c. Hence, the *challenge* is to solve this problem in a way that these continuous environment changes do not interfere with the quality of the solution. In other words, we want a Planning and Scheduling system that is *adaptable* to the changes of the environment, while preserving its *efficiency*.

Problem Representation

In our research we assume that the satellites are agile. In other words, when passing over an area that we desire to image, a satellite has many options to choose from, regarding the angle in which the imager will tilt to capture an image. We represent the problem with a directed graph, which will form the common environment the ants will traverse and update to find a solution. In the graph:

- Each *Node* represents an *pass over the AoI*
- Each *Edge* represents a *roll angle* the imager can be tilt.

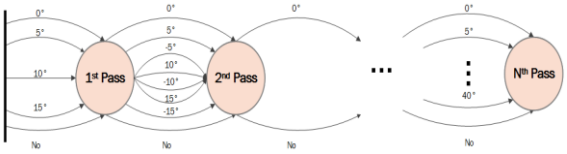
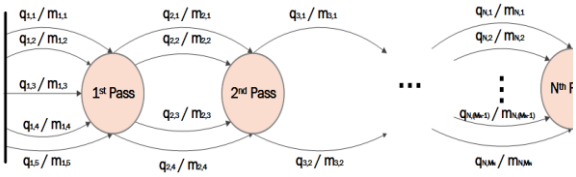


Figure 1.1. Problem representation using a directed graph.

The nodes are put in a chronological order, as the orbit of the satellite dictates. Each node is now connected to the next one with an arbitrary number of edges, each one representing an option of angle the imager will be tilt, and one representing the option of not taking a picture in this pass. Hence, each *path* starting from the first node, until the last one represents a *sequence of choices* of angles in which the imager should be tilt for each pass or a *schedule* that the satellite follows to complete the task of imaging the AoI. In order to quantify the differences among the strips available to be imaged, *quality values* (q_{ij}) are assigned to the edges, as functions of the area that they cover and the distortion of the image. Each strip is assumed to have a different quality and consume a certain amount of memory (m_{ij}). We assume to have a limited total on-board memory, M, that is only renewed when passing over a Ground Station. Formulating the problem, we assume the directed graph $G = (V,E)$, where

- V = set of Nodes
- $E = \{E_1,E_2,..,E_N\}$ and E_i = set of incoming edges in Node i.



Our goal is to find the path that visits all the Nodes and maximizes the objective function:

$$f = \sum_{i=1}^{N} q_{ij},$$

$$j = \{edges\ that\ belong\ in\ a\ selected\ Path\}$$

Under the memory constraints:

$$\sum_{i=1}^{N} m_{ij} \leq M,$$

$$M = total\ on\ board\ memory$$

Ant Colony Optimization

Ant Colony Optimization (ACO) meta-heuristic is a *probabilistic* algorithm used to find the solution in Computer Science and Operations fields' problems that can be reduced to finding optimal paths in graphs [M. Dorigo et al.] The method is summarized below:

When the ants are searching for food, in the natural world, they first wander randomly. After finding a source of food, they return to their colony but lay down pheromone trails in the path that they follow. If other ants while also looking for food find such a trail, they will probably not continue their wandering, but follow the trail instead. In case it leads them to food, they will also reinforce it when they return to their colony. The pheromone trails, however, start to evaporate over time. Hence, the longer it takes for an ant to travel back to the colony through the path it chose, the more the pheromones will evaporate. Hence, with this mechanism the amount of pheromone will become higher on the shorter paths than the longer ones, since a short path will get marched over more frequently.

The pheromone deposition helps the colony converge to an optimal solution. On the other hand, the pheromone evaporation is a means of helping the colony avoid convergence to a *locally* optimal solution. Were there no evaporation rate of the pheromone, the following ants would be more likely to choose the paths chosen by the first ones. This fact shows the importance of exploring for a sufficiently long time period and then converge to a solution. This way, the technique has higher chances of being successful.

In our case, we model the problem as a graph, and search for the path that optimizes a specific objective function. Artificial ants will run through the graph and find the desired path. Since this behaviour is inspired by nature and real ants, the artificial ants will they lay pheromone on the edges of the graph and they choose their next step with respect to a probabilistic function of the previously laid pheromone, by ants that have already traversed the graph.

$$P(e_i) = \frac{g(pheromone\ in\ e_i)}{g(total\ amount\ of\ pheromone)}$$

where $P(e_i)$ is the probability of choosing edge i, (e_i), and g is a function of the pheromone amount.

This is an indirect way of *communication* that aims at enhancing the environment (graph) with information about the quality of path components. This mechanism will lead following ants to the shortest path. The amount of pheromone an ant deposits depends on the quality of the path, which is evaluated by the objective function. This is a way to give feedback of the quality of the path an ant constructed.

ACO verification using a dynamical system

The ACO behaviour has been modelled using Ordinary Differential Equations previously, by Gutjahr [W. J. Gutjahr]. He studied the convergence speed of a number of problem representations using ODE. However, analyses are usually directed to specific algorithms, including no stability analysis. In our case, we aim at identifying the conditions of convergence. In order to apply the ACO technique to our problem, we need to understand and describe the dynamics of the long-term behaviours of the ACO algorithm. This translates to the study of under which conditions the ACO technique has the property of convergence in a solution. Hence, we are interested in understanding which is the long-term behaviour that characterises the system. We can expect numerous possible pheromone distributions, but we have a solution to our optimization problem when the system converges to one path. In this section we will present the analytical model, for the basic problem size of *1* node and *M* incoming edges. This translates to a choice among *M* roll angles in a single pass. In this way, we can easily show the basic structure of the system's dynamics by looking into its phase portrait, and thus have a deeper understanding of how the system will behave when more nodes are added. The $M-$dimensional dynamical system is:

$$\dot{\tau_1} = -\rho\tau_1 + kP_1$$
$$\dot{\tau_2} = -\rho\tau_2 + kP_2$$
$$\vdots$$
$$\dot{\tau_M} = -\rho\tau_M + kP_M$$

Where:

- τ_i = amount of pheromone in edge i
- ρ = pheromone evaporation rate
- k = amount of pheromone deposited
- $Pi = \frac{\tau_i^\alpha}{\sum \tau_j^\alpha}$, probabilistic rule based on which an edge will be chosen by an ant

We study the stability of the system using Nonlinear Dynamical Systems Theory. First, we calculate the number of equilibrium points it has, their analytical form and then define the stability of each. The results are summarized in the following Table.

	α : [0, 1)	**α= 1**	**α : (1, ∞]**
Exploration	High	Medium	Low
Convergence	Low	Medium	High

By the study of this system, we concluded that there is a critical parameter that controls the stability of the system, which is α. In the table we identify three main system's behaviors and highlight that none of them is perfect in terms of optimization. Thanks to these insight, a novel algorithm was developed in [C. Iacopino] that *combines* these behaviors to exploit their benefits. This algorithm is capable of regulating the trade-off of exploration vs convergence by oscillating alpha between the two areas.

Self – organising software system

In order to test the algorithm, a self contained component of software was designed, which wraps the entire system. It is written in Java and incorporates a fully open-source discrete-event agent-based modelling framework called MASON. The system's parameters and the objective function are configurable. The planning problems are passed in input as lists of imaging opportunities with their quality values and consumed memory while the output is a list of solutions containing the set of planned tasks. Each entity of the problem is assumed to be one agent. They are divided in three main categories: Environment agents representing the graph, Ant agents representing the computational units that update the graph and check the memory constraints and a Master agent that checks the convergence of the colony and the evaluation of each path produced.

We are considering the case of a single spacecraft. The planning problem is represented in a graph in each test, and the ant agents find the best sequence of actions.

Simulations results

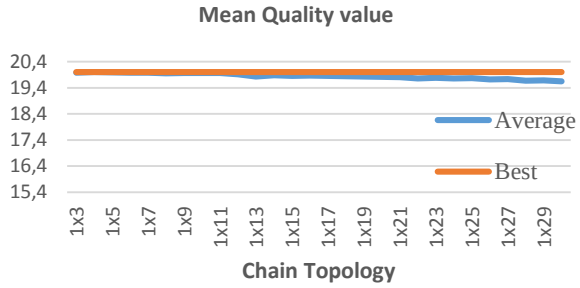
We test the *efficiency* and the *scalability* of the system. The metrics to quantify the performance of the system in these two fields will be:

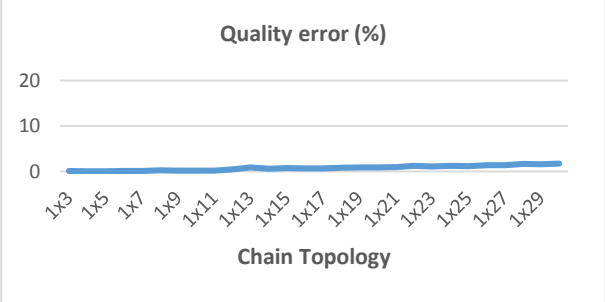
- **Quality value**. We compare the system's solution to the one given by a deterministic algorithm, performing an exhaustive search, and compute the error.
- **Convergence Time**. We measure the number of ants it takes to converge to a solution for different graph topologies.
- **Computation Time**. We check the change of the computation time, when increasing the dimensions of the problem.

We perform two types of tests. First we assume a single Node and increase the number of incoming Edges. This corresponds to increasing the number of roll angle choices in one pass. Next, we fix the number of incoming Edges to 3 and increase the number of Nodes. This corresponds to having 3 roll angles to choose from in each pass, but increasing the number of passes. We note that in this type of test, each time we add a Node, we triple the search space.

Efficiency tests

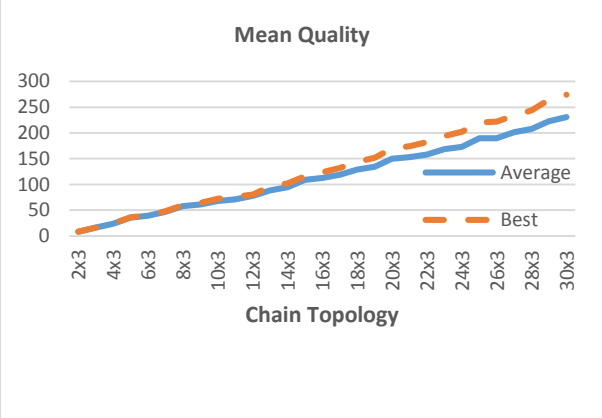
Increasing the Edges

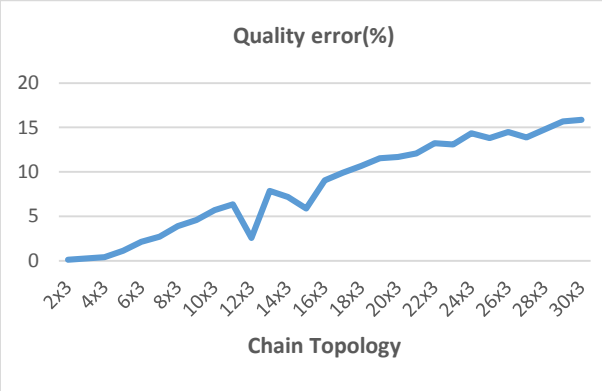




The above results show that the system is highly efficient to the increase of Edges in the graph. The error between the system's output and the best solution is up to *1.75%* with respect to the best, when having 30 Edges, or 30 roll angle choices.

Increasing the Nodes

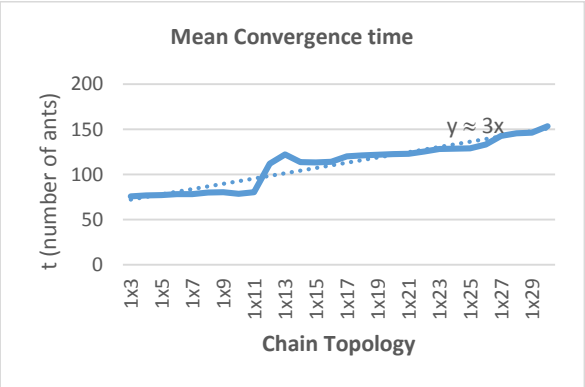


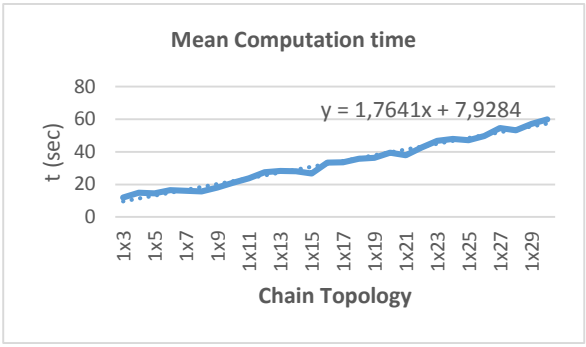


When increasing the number of Nodes, the system becomes less efficient, with the error being up to 15.8%. That is due to the fact that each Node addition results in a triple search space.

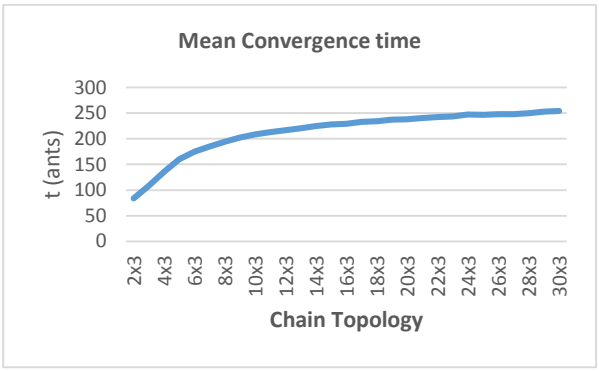
Scalability tests

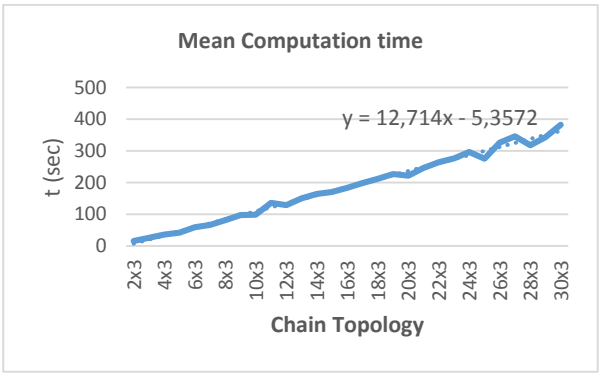
Increasing the Edges





Increasing the Nodes





The Scalability tests, for both types of tests are very encouraging. The system's convergence time increases either linearly, or logarithmically. Also, the computation time is increasing linearly, making the system *very scalable*.

Future work

SSTL Case Study

The first case study we will consider in this research is the Disaster Monitor Constellation (DMC3) produced by Surrey Satellite Technology Ltd. It is an Earth Observation mission which was launched in July 2015 and is currently in commissioning phase. The platform consists of 3 agile Earth Observation satellites at 1 m resolution.

They can change their attitude up to 45° off-nadir pointing in pitch and roll axes. This platform is the first Earth Observation constellation of low cost small satellites. It provides daily images for a wide range of applications, commercial or of public interest including disaster monitoring. This constellation offers multispectral imagery, wide swath (600km), 32m ground sample distance (GSD) and 4m panchromatic (PAN) resolution. Currently SSTL are given requests to image certain areas of the globe, and their operators manually determine how best to achieve this. We aim at using realistic data from this mission in order to test the tool that we designed. Furthermore, our goal is to integrate our method as one of SSTL's Mission Planning Systems.

ESA Case Study

During the current Ph.D. we aim at integrating the single and multiple objectives methods, in ESA's missions that employ agile constellations. A great Case Study would be the European Data Relay System (EDRS). It is a planned European constellation of state of the art GEO satellites that will relay information and data between satellites, spacecraft, UAVs, and ground stations. Given the complexity of the system, the scheduling of these activities would certainly be better performed if more than one objectives were able to be modelled and optimized. The trade-off among such objectives would definitely give an insight on the management of a system with such high level of complexity.

Multi Objective Optimization

In the *Motivation and Scope* section we stated the need for employing Multi-Objective optimization techniques to design MPS that are more adaptable and applied to a wider range of missions. We now need to understand the system's characteristics that make one method more efficient than another, in order to decide which one we are going to include in our research. The visualization of the Pareto front seems like a more desirable way to solve the problem. It allows for the trade-off between each of the objectives to appear. The problem representation that we have employed allows for many additions to the algorithm. For example, having two values of quality per edge, one for each of the two objectives we have, can be very easily integrated and will result in the Pareto front visualization, with the use of just a little more system memory. Nevertheless, this is a decision we still have to make, comparing all the advantages and disadvantages that each method carries.

Coordination Mechanism

When it comes to having multiple spacecraft collaborating to achieve a task, without communicating with each other, or having an external central controller, the coordination needs to take place by means of stigmergy. The spacecraft will share a common environment, the graph. All the possible strips now need to be represented in the graph, in order for the ants to find the path that optimizes the shared objective function. The cooperation needs to take place using the pheromone trails in the environment that all the satellites will share.

Acknowledgments

This work is co-funded by the Surrey Space Centre (SSC) of the University of Surrey, the Surrey Satellite Technology Ltd (SSTL) and the Operations Centre of the European Space Agency (ESA/ESOC).

References

N. Muscettola, P. Nayak, B. Pell, B. C. Williams, 1998. *Remote Agent: to boldly go where no AI system has gone before*, Artificial Intelligence v103, # 1-2, pp. 5–47.

T. A. Wagner et al., 2004. *An Application Science for Multi – Agent systems*.

G. Richardson, Dr K. Schmitt, M. Covert, C. Rogers, 2015. *Small Satellite Trends 2009-2013*, 29th Annual AIAA/USU Conference in Small Satellites.

E. Buchen, 2015. *Small Satellite Market Observations*, 29th Annual AIAA/USU Conference in Small Satellites.

S. A. Chien, M. Johnston, J. Frank, M. Giuliano, A. Kavelaars, C. Lenzen and N. Policella, 2012. *A generalized timeline representation, services, and interface for automating space mission operations*. In Proceedings of the 12th International Conference on Space Operations, SpaceOps.

C. Iacopino, P. Palmer, N. Policella, and A. Donati, 2014. *Planning the GENSO Ground Station Network via an Ant Colony-based approach*, In The 13th International Conference on Space Operations,
SpaceOps 2014. Pasadena, California.

M. Dorigo, T. Stuzle, 2004. *Ant Colony Optimization*, MIT Press.

W. J. Gutjahr, 2008. *First steps to the runtime complexity analysis of ant colony optimization*. Computers & Operations Research, 35(9): 2711–2727.

C. Iacopino, 2013. *Automated Planning & Scheduling for Earth Observation Constellations: an Ant Colony Approach*, PhD Thesis, University of Surrey.

Solver Paramter Tuning and Runtime Predictions of Flexible Hybrid Mathematical models.

Michael Barry
University of Basel / HES-SO
m.barry@unibas.ch / michael.barry@hevs.ch
Universitt Basel Petersplatz 1, 4001 Basel / Techno-Ple 1, 3960 Sierre
Switzerland

Abstract

In this research we consider the problems faced when using hybrid mathematical models to solve optimisation models. Such models can be configured to have different structures and can exert different behaviour and therefore can have a volatile search space, making runtime predictions and solver tuning a more complex problem. We propose an optimization configuration method that exploits the hybrid mathematical model structure for solver parameter tuning and runtime prediction.

Optimisation problems is active field of research and has been approached from several research communities, including the Artificial intelligence community as well as operations research community, each developing their own methods. All communities have had a steady contribution, yet most commonly compete rather than collaborate. Partly due to this rift and partially due to a difference on a conceptual level, many methods used in operation research have a distinct lack of influence from the Artificial intelligence or the wider computer science field of research despite being used heavily in industry. As a result, issues with maintenance and deployability are all to common and could be addressed by lessons learnt in the history of computer science.

Furthermore, these problems lead to highly inflexible models. As an example from industry, mathematical models of Hydro power stations are the dominant tool for planning their operation, optimising the return in profit based on the market prices (1) (5). This problem is well understood and any company operating such power stations use a mathematical model. Yet it has been noted that modifying these highly specialised models to new market environments have become difficult and many even choose to redevelop the model instead.

The inherent problem is the design of the models. Figure 1 dissects the common methods in operations research for solving an optimisation problem such as the Hydro power problem mentioned above into its separate components. The solvers are now considered to be separate from the modelling language, allowing a single model to be easily tested by different solvers. Software such as the general algebraic modelling system (GAMS) allows a user to develop a model in a single language and let it be solved by several different solvers. Similarly the input to the model is interchangeable, allowing a user to use data from, for example, a different year.

However, the model structure itself is considered to be static and never changing. More modern Hybrid models may incorporate a more flexible design and may have the potential to alter the actual behaviour of a model. For example, a more flexible Hydro power plant model could not be specific to just one existing plant, but could be flexible allowing a user to tailor it to any existing plant. This would require the model to simulate different types of turbines, each with different functions to define its behaviour. However, a model with changing behaviour presents a new problem for any methods for configuring the solver or for predicting the runtime of the solver.

Current methods in configuration focus on optimizing a solvers parameters for a specific problem. In such a case, the parameters are expected to improve the solvers efficiency in average over a selection of different input scenarios. However, if more flexible models are being developed that also change the behaviour of the model, solvers would show a more drastic change in performance for different scenarios. Therefore the configuration of the solvers parameters should take into consideration the design of the model itself, allowing the system to adjust the solvers parameters based on how the models behaviour has been configured. Similarly, the prediction of the solvers runtime becomes more complex. Changing the behaviour of the model also changes the complexity of the problem greatly, increasing both the need and the difficulty of an accurate prediction.

Therefore, the following three research questions arise.

- How can a mathematical model be implemented to allow a large degree of flexibility in it's behaviour?
- How can the parameters best be configured for a flexible and dynamic mathematical model?
- How can the runtime of a solver be accurately predicted for a flexible mathematical model?

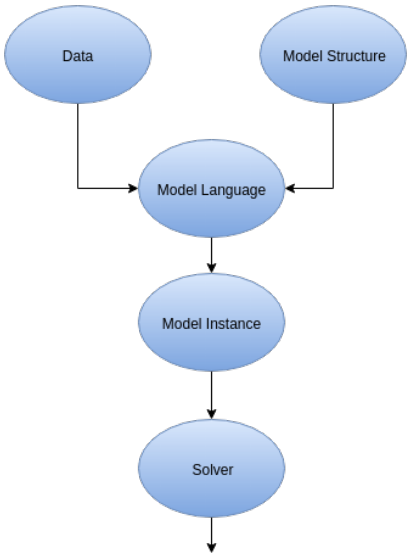


Figure 1: Graphical representation of how a problem is solved using a mathematical language and solver. Data and a model structure is input using a model language, creating a model instance. This instance can then be solved by a solver and the results are returned.

Proposed Method

Implementing software in a way that allows continues development and maintenance is well understood in computer science. One tool used to prevent it is Object Orientated design, which splits the system into several classes (module) and allows each module address another aspect of the software. Modular approaches to mathematical modelling has been used before and many new modelling languages being developed use this concept. However, this is a strong contradiction to the standard in industry, where the more classical modelling languages dominate the field. For a successful adaptation of a modular design, the concept must be usable with classical modelling languages such as GAMS. Only once a modular approach is more widely accepted will new modelling language that assists in a modular approach be more widely accepted. It also eases the integration of new models with currently used models.

A modular approach allows each aspect to be easily replaced by reimplementing the specific class. This concept can also be used in mathematical models, by separating each component into separate modules. As an example for the Hydro power plant model, it can be separated into modules each presenting an aspect of the model, such as the intraday market, a type of turbine or an environmental constraint. How this is achieved in a mathematical language is demonstrated in Figure 1 and Listing 2.

Listing 1 shows the implementation of a module containing a function describing the storage level in a Hydro power station. A further constraint on the storage variable is contained in a separate module, described in the StorageMax.gms file shown in Listing 2. By simply importing the file storageMax.gms, the module and its constraint is included in the model. A main file containing these import statements can then be used to easily modify the structure of the model by simply including or excluding modules.

Listing 1: Storage.gms

```
Equation
storage (i)
;

storage(i)..

storage(i)          =e=
 storage(i-1)
+ inflow(i)
 - release(i);

$import storageMax.gms
```

Listing 2: Storage.gms

```
Equation
storage (i)
;

storage(i)..

storage(i)          =l=          max_storage;
```

The same principle can be used for aggregate functions as shown in Listing 3 to 5 and for several versions of the same constraint as shown in Listing 6.

Listing 3: Income.gms

```
Equation
income_total (i)
;

income_total(i)..

income_total(i)          =e=
sum(m, income(i,m))
```

Listing 4: Market1.gms

```
Equation
income(i)
;

income(i)..
```

```
income(i)          =e=
price(i,1) * production(i,1)
```

Listing 5: Market2.gms

```
Equation
income(i)
;

income(i)..

income(i)          =e=
price(i,2) * production(i,2)
```

As shown in Listing 3, the total income is calculated by summing the income from each market. The income for each market is calculated in separate modules. By including or excluding the module market1 and market2, the market can be added or removed.

Listing 6: storageLoss.gms

```
Equation
storage (i)
;

storage(i)..

storage(i)          =e=
storage(i-1)
+ inflow(i)
- sum(m, release(i,m))
- loss(i)
```

Listing 6 shows an alternative implementation of the storage module described in Listing 1 and can be used to replace the previous implementation. Altogether, these methods can be used to create a model, which allows its behaviour to be modified by simple import statements.

Instances off each module can be used to model multiple occurrences of physical objects. For example, there may be different modules for different type of turbines, such as a Pelton turbine or a Francis turbine. However, a Hydro power station may contain several turbines of the same type. Therefore several instances of a turbine may exist. The concept of instantiation is fairly simple though and can be implemented in a functional language through a set of functions and a table containing the parameters for each instance. In a similar manor, it is possible for a mathematical language.

Modules depend on others and some common sense restrictions must be respected. For example, the Hydro power model must contain a type of turbine to allow power production and a market must be available to sell the energy produced. However, it is also possible to operate on several markets at the same time. Similarly, modules that implement the same aspect and therefore the same constraint but in a different may can only be selected once within a model, effectively creating an *exclusive or* relationship. Instances themselves are subject to restrictions, as a reservoir must be connected to a turbine or river to allow water to exit again to avoid flooding. The relationship between different modules and different instances can be captured in predicate logic and used as a validation method to determine a models validity before passing it to the solver.

As this approach combines a top down model, consisting of the mathematical model, and a bottom up model model, consisting of the validity logic, it would be considered a Hybrid model within the Operations Research community (4). Within the computer science community, this method is conceptually different and would be recognised as dependency injection or inversion of control, as the module with the desired behaviour is selected at runtime. Despite different ways of looking at it, such a design is understood and accepted by both communities.

The benefit of such an implementation is similar to those of Object orientated design, including reusability and maintainability. In addition, a new developer does not require knowledge of the entire system to update one aspect of the system. Furthermore, a model can then be configured by selecting which modules to use in a simulation (3) (2). This can allow a model to behave differently in each configuration as shown in Figure 2.

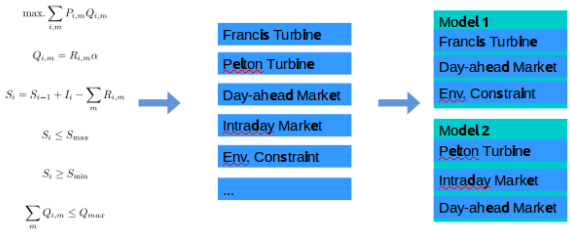


Figure 2: Graphical representation of how modules can be combined to create different configurations with different behaviours. The functions defining the model are separated into modules, which are then combined to create a model configuration.

The ability to configure the behaviour of a model sets it apart from most other model designs. Such functionality is sometime implemented to some extent utilising simple *if* statements throughout the model, such as shown in Listing 7.

Listing 7: storageLoss.gms

```
Equation
storage (i)
;

storage(i)..
```

```
storage(i)          =e=

storage(i-1)
+ inflow(i)
- sum(m, release(i,m))
- loss(i)$(include eq 1)
```

However, it is limited. Extensive use of such statements result in the model being riddled with if statements making it unreadable. In addition, each flexible aspect must be implemented manually rather than being inherently flexible by design. By using a modular approach instead it separates this logic from the functions, adding a layer of abstraction.

The resulting model would result in a large degree of flexibility, but will also result in varying complexity. Based on which and how many modules are selected at runtime, the runtime of the solver can vary greatly. Predicting the runtime of the solver can be done using machine learning methods (6). However, predicting the runtime of the solver becomes more difficult when using a model with changing behaviour. In a standard model, the search space is relatively stable compared to the proposed model. Different inputs will only modify the search space to some degree, while changing the the number of constraints and the constraints themselves changes the search space drastically.

However, there is some knowledge that can be extracted from the model structure itself and can then be used to more accurately predict the runtime. Simply using the number of modules can already help in estimating the complexity of the model as shown in Figure 3 and further information on how each module depends on the other and the validity logic contains useful knowledge of the models structure. Overall, there are many candidate features which may prove to be a good indication of the models runtime.

Using methods from machine learning such as a correlation matrix and a neural network, the best features could be selected and their relationship to the models runtime could be learned. Depending on which features are the most viable, this method could potentially also be viable for more modules or even entire models that the neural network has not been trained on. This could possibly be used as a global tool rather than being limited to a specific model. The features to be selected and their relationship to the runtime must be examined in a set of experiments. Additionally, how well the relationship can be learned and the extent to which this method can be used for unknown modules must also be investigated.

Most solvers such as the IBM CPLEX solver have several parameters that can be fine tuned by the user to boost the solvers performance for a specific problem. Parameters can modify the heuristics, probing, cutting and much more (7). Automatic configuration of these parameters has been studied to a great extent (7) and is even included in some solvers functionality. However, they do not considered a flexible model structure.

Therefore, a tuning process would have to be applied each time the structure changes. Again, knowledge from the models structure could be used to help tune the parameters for a specific structure. How much the optimal parameters for each structure vary, the performance increase and which parameters to modify must be tested in a set of experiments. Additionally, the correlation between models structure and the parameters as well as how well this relationship can be learned will have to be investigated.

The resulting system would allow a model to be configured for many different scenarios and can generate an ideal parameter configuration and runtime prediction for that scenario before being executed.

Experiments

A fully functioning prototype demonstrating the feasibility and the benefits of a model developed though a modular design will be developed. As a case study, a model of a Hydro power station is developed that has the flexibility to simulate several market and investment scenarios. Due to its flexibility, it will not be tied to a specific power plant but rather be general and can be configured for any hydro power plant. It is hoped that such a model can be used as a example for modular design and can be continuously updated and integrated with other models.

The results of the previously mentioned experiments will demonstrate how such a design can assist in predicting the runtime of a flexible model. In addition, they will determine which features of the model structure is best suited for an accurate prediction for the solvers runtime. Knowledge can be extracted from the modules themselves, such as in the most simple case the number of functions within the model, as well as their relationship with other modules, such as how many variables are shared between the modules. It is hoped that a set of features can be selected that are easily extracted from a model to allow this method be applied to modules or entire models that are not contained in the training set.

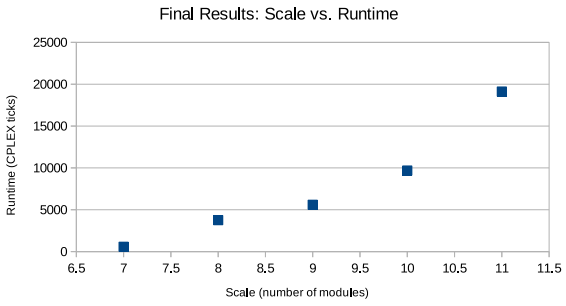


Figure 3: Initial results of running different model configurations. It shows how a simple parameter such as the number of modules has a correlation to the runtime of the model.

Therefore, it would allow this method be used on any model that follows the same design pattern. The success of this experiment relies on if general features for each module can be used, or whether the system has to learn the impact that each module has on the complexity of the model. If the latter is the case, the system would have to train on

each new module that was added before making a prediction. Although still useful, it would limit its reusability for other models.

The experiments concerning the parameter tuning of the solver will determine which parameters should be tuned for different structures to produce a performance boost. As a first step, the experiments must determine whether different configurations of the model require different parameter configurations. If this is the case, a performance boost is viable. In this case, experiments will determine which parameters are best suited for configuration and to what extent the relationship between structure configuration and optimal parameter configuration can be learned.

Evaluation

The evaluation of these methods are important in demonstrating their viability. It allows comparison to existing methods and is important for determining whether it is not only novel, but also useful to the research community.

The success of the model design will be evaluated in two case studies involving a hydro power station model. It will be compared to the current state of the art in industry and will also be evaluated on the bases of what further functionality is achieved through a modular design. The extent of its flexibility, such as whether it can simulate different type of Hydro power plants, whether it can be used to investigate investment opportunities and how well it can compete with other models in terms of accuracy in real scenarios will be tested.

The runtime prediction can be tested for accuracy by comparing the predicted runtime with actual runtime prediction. In addition, a comparison can be made with prediction methods that work with a static model structure. Although an unfair comparison, it can show the feasibility of predicting the runtime for flexible models.

The methods used for tuning the solvers parameters for different structures can be evaluated by comparing it to a base case parameter configuration. This base case can be created by choosing a configuration that is optimal in average over different model structures. Through a comparison of the solvers runtime to such a base case, the benefit of using methods based on the models structure can be measured.

Contributions

Successful investigation of the previously stated research questions would result in a fully functional system, that is adaptable, automatically configurable and predictable. However, apart from the system itself, it is expected that several scientific insights are gained. It will extend our model development methods, our knowledge of automatic parameter tuning and our understanding of what elements in a model affects its complexity.

The model design will help the efforts to transition form classical modelling methods to more agile methods similar to what is commonly used now in computer science as well as reduce the gap between the computer science and operations research communities. The model itself will be useful as an experiment base allowing the study of how small changes to a module affects its complexity. This understanding can then be used to better understand the runtime of our solvers and how best to tune the solver to minimise these effects.

Conclusions

In conclusion, a change in design of mathematical model both requires and assists in developing methods for automatic solver tuning and runtime predictions. Current methods are limited by the assumption that the structure remains static and the search space only shifts due to different inputs, rather than different model behaviours. Changing behaviours makes the tuning of the solver and the prediction of its performance more difficult, but may also deepen our understanding of this relationship, further improving the state of the art.

Acknowledgements

This work has been done in the context of the SNSF funded project *Hydro Power Operation and Economic Performance in a Changing Market Environment*. The project is part of the National Research Programme *Energy Transition* (NRP70).

References

[1] Alfieri, L.; Perona, P.; and Burlando, P. 2006. Optimal water allocation for an alpine hydropower system under changing scenarios. *Water resources management* 20(5):761–778.

[2] Barry, M., and Schumann, R. 2015. Dynamic and configurable mathematical modelling of a hydropower plant research in progress paper. In *Presented at the 29. Workshop "Planen, Scheduling und Konfigurieren, Entwerfen" (PuK 2015)*.

[3] Barry, M.; Schillinger, M.; Weigt, H.; and Schumann, R. 2015. Configuration of hydro power plant mathematical models. In *Energy Informatics: Proceedings of the Energieinformatik 2015*, volume 9424 of *Lecture Notes in Computer Sciences*. Springer.

[4] Böhringer, C. 1998. The synthesis of bottom-up and top-down in energy policy modeling. *Energy economics* 20(3):233–248.

[5] Guo, S.; Chen, J.; Li, Y.; Liu, P.; and Li, T. 2011. Joint operation of the multi-reservoir system of the three gorges and the qingjiang cascade reservoirs. *Energies* 4(7):1036–1050.

[6] Hutter, F.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206:79–111.

[7] Klotz, E., and Newman, A. M. 2013. Practical guidelines for solving difficult mixed integer linear programs. *Surveys in Operations Research and Management Science* 18(1):18–32.

Thesis Abstract: Constructing Heuristics for PDDL+ Planning Domains

Wiktor Piotrowski
Supervised by: **Daniele Magazzeni** and **Maria Fox**
Department of Informatics
King's College London
United Kingdom

Abstract

Planning with hybrid domains modelled in PDDL+ has been gaining research interest in the Automated Planning community in recent years. Hybrid domain models capture a more accurate representation of real world problems, that involve continuous processes, than is possible using discrete systems. However, solving problems represented as PDDL+ domains is very challenging due to the construction of complex system dynamics, including non-linear processes and events, and vast search spaces.

The main focus of my PhD is to mitigate these challenges by developing domain-independent heuristics for planning in hybrid domains modelled in PDDL+. This is a very real issue as only a handful of planners can cope with hybrid domains and, fewer still with the full set of PDDL+ features and non-linear behaviour.

1 Introduction

Over the years, Automated Planning research has been continuously attempting to solve the most advanced and complex planning problems. The standard modelling language, PDDL (McDermott et al. 1998), has been evolving to accommodate new concepts and operations, enabling research to tackle problems more accurately representing real-world scenarios. Recent versions of the language, PDDL2.1 and PDDL+ (Fox and Long 2003; 2006), enabled the most accurate standardised way yet, of defining hybrid problems as planning domains.

Planning with PDDL+ has been gaining research interest in the Automated Planning community in recent years. Hybrid domains are models of systems which exhibit both continuous and discrete behaviour. They are amongst the most advanced models of systems and the resulting problems are notoriously difficult for planners to cope with due to non-linear behaviour and immense search spaces.

My research aims mitigate these issues by developing domain-independent heuristics able to reason with nonlinear system dynamics and PDDL+ features such as processes and events. These heuristics are being implemented in UPMurphi as a proof of concept.

We begin by outlining the related work done in the area of hybrid domains and PDDL+ planning in section 2. We discuss the relevance of the research problem and motivation for tackling it in section 3. Section 4 describes our methodology for dealing with hybrid domains. We then outline the contribution made and ongoing research in section 5. Section 6 describes the future research. Section 7 concludes the thesis summary.

2 Related Work

Various techniques and tools have been proposed to deal with hybrid domains (Penberthy and Weld 1994; McDermott 2003; Li and Williams 2008; Coles et al. 2012; Shin and Davis 2005). Nevertheless, none of these approaches are able to handle the full set of PDDL+ features, namely non-linear domains with processes and events. More recent approaches in this direction have been proposed by (Bogomolov et al. 2014), where the close relationship between hybrid planning domains and hybrid automata is explored. (Bryce et al. 2015) use dReach with a SMT solver to handle hybrid domains. However, dReach does not use PDDL+, and cannot handle exogenous events.

On the other hand, many works have been proposed in the model checking and control communities to handle hybrid systems. Some examples include (Cimatti et al. 2015; Cavada et al. 2014; Tabuada, Pappas, and Lima 2002; Maly et al. 2013), sampling-based planners (Karaman et al. 2011; Lahijanian, Kavraki, and Vardi 2014). Another related direction is *falsification* of hybrid systems (i.e., guiding the search towards the error states, that can be easily cast as a planning problem) (Plaku, Kavraki, and Vardi 2013; Cimatti et al. 1997). However, while all these works aim to address a similar problem, they cannot be used to handle PDDL+ models. Some recent works (Bogomolov et al. 2014; 2015) are trying to define a formal translation between PDDL+ and standard hybrid automata, but so far only an over-approximation has been defined, that allows the use of those tools only for proving plan non-existence.

To date, the only viable approach in this direction is PDDL+ planning via *discretisation*. UPMurphi (Della Penna, Magazzeni, and Mercorio 2012), which implements the discretise and validate approach, is able to deal with the full range of PDDL+ features. The main drawback of UPMurphi is the lack of heuristics, and this strongly limits its

scalability. However, UPMurphi was successfully used in the multiple-battery management domain (Fox, Long, and Magazzeni 2012), and more recently for urban traffic control (Vallati et al. 2016). In both cases, a domain-*specific* heuristic was used.

3 Problem Statement & Motivation

Automated Planning is a crucial part of a multitude of systems in almost every domain of science and technology. However, in certain cases the complexity or scale of the systems has outgrown the capabilities of the modelling language rendering the planning domain either too inaccurate or too cumbersome to express.

When first introduced, PDDL+ allowed new, more complex problems, closely resembling real-world scenarios, to be modelled. Though planning is now able to express complex hybrid domains, solving these problems is very challenging due to nonlinear behaviour, state explosion and continuous variables rendering the reachability problem undecidable. As described in Section 2, various planning tools using different approaches have been developed over the past years to tackle problems set in hybrid domains. However, the vast majority cannot deal with the full set of PDDL+ features and/or nonlinear behaviour. This significantly limits the relevance of Automated Planning for a wide range of applications since only restricted and/or downscaled hybrid models can be handled. As a result, some classes of planning problems, relevant to today's science and technology, are being solved using domain-specific heuristics or approaches from outside planning altogether (Mixed-Integer Programming, Genetic Algorithms, etc.).

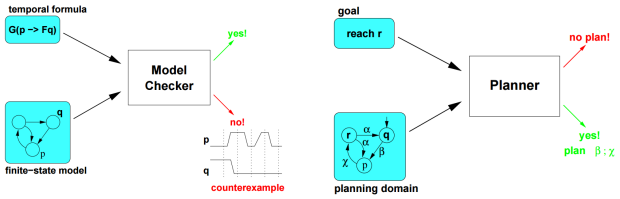
In addition to the inability to reason with some PDDL+ features and/or non-linearity, current planning tools scale poorly when presented with larger problem instances. This is due to the absence or poor performance of heuristics in the presence of vast search space, exogenous processes and events, and non-linearity. Currently, heuristics applied to hybrid systems have either been developed for a different subclass of problems (e.g. PDDL2.1), or to reason with only a subset of the features expressible in PDDL+.

The main motivation of this research is to advance PDDL+ planning to tackle larger and more complex problems by addressing the apparent lack of efficient domain-independent heuristics devised specifically for hybrid domains. To significantly increase the performance of planners in hybrid domains, heuristics should be designed to directly reason with the complex system dynamics, and PDDL+ features, i.e. processes and events.

4 Methodology

Reasoning with PDDL+ features and complex, often nonlinear, system dynamics is a challenging objective for Automated Planning tools. As shown in Section 2, all current approaches have drawbacks significantly limiting their performance and capabilities, often rendering them inadequate for the complex problems at hand.

Our approach of coping with PDDL+ domains combines two successful paradigms, ***planning as model checking*** and



(a) Model Checking (b) Planning

Figure 1: Similarity of Model Checking and Automated Planning

Discretise & Validate.

Planning as model checking (Cimatti et al. 1997; Bogomolov et al. 2014) is an approach applying model checking methods to finding the goal state in Automated Planning. Model checking and planning have striking similarities meaning that methods from one field can be exploited to improve performance in the other field. Searching for a goal in planning (Fig.1a) can be seen as searching for an error state in model checking(Fig.1b). Analogously, the error trace in model checking corresponds to a trajectory to the goal state in planning. Planning as model checking has been successfully used in various scenarios, and is gaining more research interest (multiple publications and workshops at top conferences).

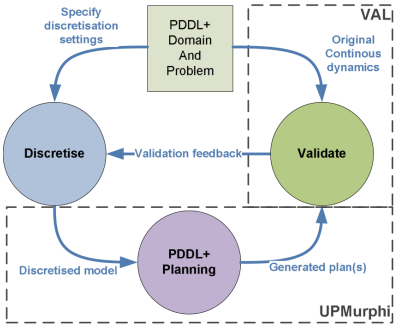


Figure 2: The Discretise & Validate process

As mentioned before, planning via discretisation is, to date, the only viable approach to PDDL+ planning. Our approach is based on the Discretise & Validate approach (Della Penna, Magazzeni, and Mercorio 2012; Della Penna et al. 2009). It approximates the continuous dynamics of systems in a discretised model with uniform time steps and step functions. Using a discretised model and a finite-time horizon ensures a finite number of states in the search for a solution. Solutions to the discretised problem are validated against the original continuous model using VAL (Howey, Long, and Fox 2004). If the plan at a certain discretisation is not valid, the discretisation can be refined and the process iterates. Discretise & Validate technique has been the basis of UPMurphi's success in the planning domain. An outline of the Discretise & Validate process is shown in Fig. 2.

5 Contributions

This section describes my contribution to date. Our development resulted in DiNo, a new heuristic planner designed for PDDL+ domains equipped with informed search algorithms and Staged Relaxed Planning Graph+ (SRPG+), a new domain-independent heuristic based on the Temporal Relaxed Planning Graph. Our publication describing DiNo is currently under review for IJCAI 2016.

5.1 DiNo

We introduced DiNo, a new planner for PDDL+ problems with mixed discrete-continuous non-linear dynamics. DiNo is built on UPMurphi, and uses the planning-as-model-checking paradigm and relies on the Discretise & Validate approach to handle continuous change and non-linearity. Though UPMurphi has been successful, it scales poorly due to exhaustive uninformed search algorithm, DiNo compensates for the lack of heuristics and shows significant improvement over its predecessor.

DiNo uses a novel relaxation-based domain-independent heuristic for PDDL+, Staged Relaxed Planning Graph+ (SRPG+). The heuristic guides the Enforced Hill-Climbing algorithm (Hoffmann and Nebel 2001). In DiNo we also exploit the deferred heuristic evaluation (Richter and Westphal 2010) for completeness (in a discretised search space with a finite horizon). States generated through non-helpful actions are considered (inserted into the queue) but they are not heuristically evaluated, instead they are assigned the heuristic value of their parent state (i.e. the are deemed no better than their parent state).

DiNo is currently the only heuristic planner capable of handling non-linear system dynamics combined with the full PDDL+ feature set.

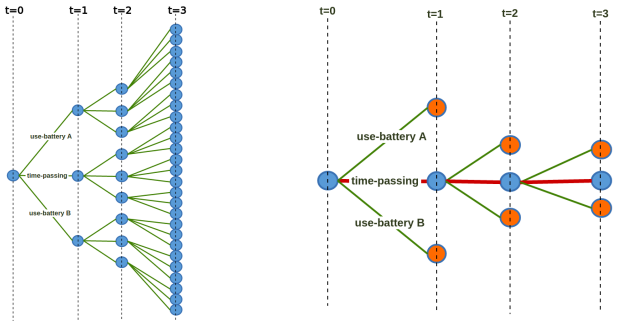
5.2 SRPG+

This section describes the Staged Relaxed Planning Graph+ (SRPG+) domain-independent heuristic designed for PDDL+ domains and implemented in DiNo.

The SRPG+ heuristic follows from Propositional (Hoffmann and Nebel 2001), Numeric (Hoffmann 2003; 2002) and Temporal RPGs (Coles et al. 2012; 2008; Coles and Coles 2013). The original problem is relaxed and does not account for the delete effects of actions on propositional facts. Numeric variables are represented as upper and lower bounds which are the theoretical highest and lowest values each variable can take at the given fact layer. Each layer is time-stamped to keep track of the time at which it occurs.

The Staged Relaxed Planning Graph+, however, extends the capability of its RPG predecessors by tracking processes and events to more accurately capture the continuous and discrete evolution of the system.

Apart from the inclusion of processes and events, the Staged RPG significantly differs from the Temporal RPG in time-handling. The SRPG explicitly represents ***every*** fact layer with the corresponding time clock, and in this sense the RPG is "staged", as the finite set of fact layers are separated by the discretised time step (Δt). In contrast, the TRPG takes time constraints into account by time-stamping each



(a) UPMurphi (b) DiNo

Figure 3: Branching of search trees (Blue states are explored, orange are visited. Red edges correspond to helpful actions)

fact layer at the earliest possible occurrence of a happening. Only fact layers where values are directly affected by actions are contained in the TRPG.

Time Passing The time-passing action plays an important role as it propagates the search in the discretised timeline. During the normal expansion of the Staged Relaxed Planning Graph, the time-passing is one of the Δ-actions and is applied at each fact layer. Time-passing can be recognised as a helpful action (Hoffmann and Nebel 2001) when its effects achieve some goal conditions (or intermediate goal facts). However, if at a time t there are no helpful actions available to the planner, time-passing is assigned highest priority and used as a helpful action. This allows the search to quickly manage states at time t where no happenings of interest are likely to occur.

This is the key innovation with respect to the standard search in the discretised timeline performed, e.g., by UPMurphi. Indeed, the main drawback of UPMurphi is in that it needs to expand the states at each time step, even during the *idle periods*, i.e., when no interesting interactions or effects can happen. Conversely, SRPG+ allows DiNo to identify time-passing as a helpful action during *idle periods* and thus advance time, mitigating the state explosions.

An illustrative example is shown in Figure 3, that compares the branching of the search in UPMurphi (Fig. 3a) and DiNo (Fig. 3b) when planning with a Solar Rover domain. The domain is described in detail in Section 5. Here we highlight that the planner can decide to use two batteries, but the goal can only be achieved thanks to a Timed Initial Literal ((Edelkamp and Hoffmann 2004)) that is triggered only late in the plan. UPMurphi has no information about the future TIL, therefore it tries to use the batteries at each time step. On the contrary, DiNo recognises the time-passing as a helpful action, and this prunes the state space dramatically.

Processes and Events The SPRG+ heuristic improves on the Temporal Relaxed Planning Graph and extends its functionality to reason with information gained from PDDL+ features, namely the processes and events.

As the SRPG+ heuristic is tailored for PDDL+ domains, it takes into account processes and events. In the SRPG,

the continuous effects of processes are handled in the same manner as durative action effects, i.e. at each action layer, the numeric variables upper and lower bounds are updated based on the time-step functions used in the discretisation to approximate the continuous dynamics of the domain.

Events are checked immediately after processes and their effects are relaxed as for the instantaneous actions. The events can be divided into "good" and "bad" categories. "Good" events aid in finding the goal whereas "bad" events either hinder or completely disallow reaching the goal. Currently, DiNo is agnostic about this distinction. However, as a direct consequence of the SRPG+ behaviour, DiNo exploits good events and ignores the bad ones. Future work will explore the possibility of inferring more information about good and bad events from the domain.

5.3 Evaluation of DiNo

In this section the performance of DiNo is evaluated on PDDL+ benchmark domains. Note that the only planner able to deal with the same class of problems is UPMurphi, which is also the most interesting competitor as it can highlight the benefits of the proposed heuristic. For sake of completeness, where possible, a comparison with other planners able to handle (sub-class of) PDDL+ features is presented, namely POPF (Coles et al. 2010; Coles and Coles 2013) and dReach (Bryce et al. 2015).

For the experimental evaluation, two benchmark domains were considered: generator and car. We also developed two further domains for the evaluation to highlight specific aspects of DiNo: Solar Rover shows how DiNo handles TILs, and Powered Descent further tests its non-linear capabilities.

Generator The domain models a diesel-powered generator which has to be refueled to run for a given duration without overflowing or running dry. We evaluate DiNo on both the linear and non-linear versions of the problem. The non-linear generator models fuel flow rate using Torricelli's Law which has been previously encoded in PDDL by (Howey and Long 2003). In both variants, we increase the number of tanks available to the planner while decreasing the initial generator fuel level for each subsequent problem.

Solar Rover We developed the Solar Rover domain to test the limits and potentially overwhelm discretisation-based planners, as finding a solution to this problem relies on a TIL that is triggered only late in the plan. The task revolves around a planetary rover transmitting data which requires a certain amount of energy. To generate enough energy the rover can choose to use its batteries or gain energy through its solar panels. The goal can only be reached through a sunshine event which is triggered by a TIL at a certain time. The set of problem instances for this domain has the trigger fact become true at an increasingly further time point (50 to 1000 time units). In the non-linear variant of the domain, the TIL triggers a process charging the rover's battery at an exponential rate.

Powered Descent We developed a new domain which models a powered spacecraft landing on a given celestial body. The vehicle gains velocity due to the force of gravity. The available action is to fire thrusters to decrease its velocity. The thrust action duration is flexible and depends on the available propellant mass. The force of thrust is calculated via Tsiolkovsky rocket equation (Turner 2008). The goal is to make a controlled landing from the initial altitude within a given time-frame. The spacecraft has been modelled after the Lunar Descent Module used in NASA's Apollo missions.

Car The Car domain (Fox and Long 2006) shows that DiNo does not perform well on all types of problems, the heuristic cannot extract enough information from the domain and as a result loses out to UPMurphi by approximately one order of magnitude. This variant of the Car domain has its overall duration and acceleration limited, and the problems are set with increasing bounds on the acceleration (corresponding to the problem number).

Due to lack of heuristic information extracted from the domain, DiNo reverts to a blind Breadth-First search and, in the end, explores the same number of states as UPMurphi. The results for the Car domain in Table 1 show the overhead generated by the SRPG+ heuristic in DiNo.

Overall, the results show that DiNo holds a significant advantage in performance over UPMurphi and other competitors in most test domains.

6 Future Research

This section describes the ongoing research and future plans for my PhD. The focus is on defining and developing new heuristics as well as promoting the PDDL+ planning in the AI community.

6.1 Temporal Pattern Database

After developing the Staged relaxed Planning Graph+, we decided to examine another successful class of heuristics - Pattern Databases (PDB). The pattern database is a look-up table indexed by a subset of the state and containing a precomputed heuristic value that reflects the cost of solving the corresponding subproblem. Each state explored during concrete search is assigned the abstract cost of its corresponding abstract state in the PDB, as the heuristic value.

The key element to a high-performing PDB is the abstraction selection. In planning, PDBs have been applied to propositional domains where the abstraction would obscure part of each state's variable set ((Edelkamp 2002; Haslum et al. 2007; Edelkamp 2014)). On the other hand, research in PDBs in model checking has concentrated on abstracting continuous variables ((Bogomolov et al. 2013)).

We build on research conducted in both fields to develop Temporal Pattern Database (TPDB), a new heuristic method with novel features enabling tackling complex problems with non-linear dynamics and full PDDL+ feature set. Our heuristic simultaneously handles full PDDL+ domains and prunes a substantial part of the search space. Processes and events are accounted for by default, their effects are automatically applied when building the TPDB.

The abstraction we devised is two fold: time abstraction and state abstraction. Combining the two abstractions manages to keep the TPDB efficient and reasonable in size.

Time abstraction is a function which increases the discretised time step (Δt) for use in the abstract state space in the

	LINEAR GENERATOR				NON-LINEAR GENERATOR		LINEAR SOLAR ROVER		NON-LINEAR SOLAR ROVER		POWERED DESCENT		CAR	
PROBLEM	**DiNo**	**POPF**	**dReach**	**UPMurphi**	**DiNo**	**UPMurphi**	**DiNo**	**UPMurphi**	**DiNo**	**UPMurphi**	**DiNo**	**UPMurphi**	**DiNo**	**UPMurphi**
1	0.34	0.01	2.87	140.50	3.62	X	0.70	203.26	1.10	288.94	0.68	0.18	1.74	0.22
2	0.40	0.01	X	X	0.78	X	0.92	X	2.58	X	1.04	0.74	4.56	0.30
3	0.50	0.05	X	X	2.86	X	1.26	X	4.74	X	1.88	2.98	8.26	0.42
4	0.60	0.41	X	X	59.62	X	1.52	X	7.10	X	3.52	7.18	10.28	0.54
5	0.74	6.25	X	X	1051.84	X	1.80	X	9.58	X	2.88	30.08	14.16	0.66
6	0.88	120.49	X	X	X	X	2.04	X	12.86	X	3.14	126.08	15.78	0.68
7	1.00	X	X	X	X	X	2.28	X	16.48	X	5.26	322.16	17.08	0.72
8	1.16	X	X	X	X	X	2.64	X	21.38	X	3.82	879.52	18.90	0.72
9	1.38	X	X	X	X	X	2.98	X	26.74	X	1.58	974.60	19.30	0.76
10	2.00	X	X	X	X	X	3.30	X	29.90	X	2.26	X	19.50	0.78
11	1.84	X	X	X	N/A	N/A	3.50	X	35.96	X	11.24	X	N/A	N/A
12	2.06	X	X	X	N/A	N/A	3.74	X	42.54	X	42.24	X	N/A	N/A
13	2.32	X	X	X	N/A	N/A	4.00	X	48.06	X	14.90	X	N/A	N/A
14	2.46	X	X	X	N/A	N/A	4.38	X	55.46	X	61.94	X	N/A	N/A
15	2.88	X	X	X	N/A	N/A	5.20	X	62.84	X	19.86	X	N/A	N/A
16	2.94	X	X	X	N/A	N/A	5.40	X	74.50	X	80.28	X	N/A	N/A
17	3.42	X	X	X	N/A	N/A	5.08	X	86.96	X	2.94	X	N/A	N/A
18	3.54	X	X	X	N/A	N/A	5.64	X	95.66	X	2234.88	X	N/A	N/A
19	3.76	X	X	X	N/A	N/A	6.12	X	102.86	X	X	X	N/A	N/A
20	4.26	X	X	X	N/A	N/A	6.02	X	117.48	X	X	X	N/A	N/A

Table 1: Run time to find a valid solution (in seconds) ("X" - planner ran out of memory, "N/A" - problem not tested)

TPDB. This allows the TPDB to store fewer states and keep its size manageable. However, choosing the correct time abstraction has to be a compromise between the size of the TPDB and the precision. Too coarse abstraction can miss the adverse events occurring between the abstract state time points and cause significant back-tracking.

State abstraction is a function which reduces the precision of states' continuous variables. This method compensates for the discrepancies between the values of continuous variables in concrete and abstract states. Due to the varied discretisation, real variables in abstract states in the TPDB (generated using the abstracted time step) can differ from their corresponding variables in concrete states (achieved using the concrete time step Δt). Choosing the precision for the state abstraction is crucial for the Temporal Pattern Database. On the one hand, choosing a coarser precision for real variables will shrink the size of the TPDB (each abstract state will correspond to a larger number of concrete states). On the other hand, choosing finer precision will make the heuristic estimates more accurate. When choosing the precision value, one should aim to balance the two aspects.

The Temporal Pattern Database is a structure which maps abstract state-action pairs to the length of the shortest trajectory to an abstract goal state. A TPDB is built by executing the applicable actions under time abstraction to generate the subsequent abstract states until a the abstract goal state is found, or the finite temporal horizon T is reached (meaning the bounded abstract problem is unsolvable and the horizon should be increased, or the discretisation refined). The abstract distances stored in the TPDB are used in the concrete search as the heuristic estimate for each considered state.

We are currently in the process of implementing the Temporal Pattern Database into UPMurphi to evaluate our concept. PDBs have proven to be a high performing approach for both model checking and propositional planning. We believe that transforming this approach to reason with PDDL+ domains can generate a very efficient and powerful heuristic.

6.2 PDDL+ Benchmarks

Conducting novel research is obviously crucial to a PhD but helping in identifying and mitigating the shortcomings of one's field of study is just as important.

One of the most pertinent inconsistencies in PDDL+ planning was the use of domains for evaluation. Using inconsistent domains for empirical evaluation makes comparison of planner performances difficult or impossible. To date, no standardised set of benchmark domains exists. Despite the Generator and Car domains being thought of as benchmark domains, multiple variants of them exist and are being used by different planning research groups around the world. We concluded that a set of benchmark PDDL+ domains needs to be readily available for the community to provide a fair, unbiased comparison with competing planners, and began compiling the test suite.

First, we collected the standard hybrid domains used in planning and model checking literature. Examples of these include Generator, Car,and Battery Management. To further expand the suite, we began developing our own PDDL+ models. The top priority for our research is to diversify the domains to account for various classes of problems that can be encoded in PDDL+.

7 Conclusion

Efficient, high-performing heuristics are an integral and essential part of Automated Planning. With the PDDL+ planning area increasingly gaining research interest, it is very important to continue developing advanced heuristics to match the requirements of emerging hybrid domains. Heuristics need to mitigate vast search spaces but also reason with complex system dynamics to estimate the evolution of the system to a satisfactory extent. On the other hand, it is crucial to continue inventing novel PDDL+ domains to further test planning tools and set up a benchmark suite to allow fair comparison between competitor planners.

We have presented the Staged Relaxed Planning Graph+, a domain-independent heuristic developed entirely for PDDL+ planning domains implemented in DiNo, the first heuristic planner capable of handling the full PDDL+ feature set and non-linear system dynamics. We have shown that reasoning directly with processes and events can produce advantages in performance of a planner.

Pattern Database heuristics proved successful in model checking and classical planning. Our current research on the Temporal Pattern Database is built upon these approaches and shows promise for complex PDDL+ domains. The immediate future will be dominated by our efforts to implement the TPDB heuristic into UPMurphi and empirically evaluate its performance on PDDL+ domains.

References

Bogomolov, S.; Donzé, A.; Frehse, G.; Grosu, R.; Johnson, T. T.; Ladan, H.; Podelski, A.; and Wehrle, M. 2013. Abstraction-based guided search for hybrid systems. In *Model Checking Software*. Springer. 117–134.

Bogomolov, S.; Magazzeni, D.; Podelski, A.; and Wehrle, M. 2014. Planning as Model Checking in Hybrid Domains. In *AAAI*. AAAI Press.

Bogomolov, S.; Magazzeni, D.; Minopoli, S.; and Wehrle, M. 2015. PDDL+ planning with hybrid automata: Foundations of translating must behavior. In *ICAPS*, 42–46.

Bryce, D.; Gao, S.; Musliner, D. J.; and Goldman, R. P. 2015. SMT-Based Nonlinear PDDL+ Planning. In *AAAI*, 3247–3253.

Cavada, R.; Cimatti, A.; Dorigatti, M.; Griggio, A.; Mariotti, A.; Micheli, A.; Mover, S.; Roveri, M.; and Tonetta, S. 2014. The nuXmv symbolic model checker. In *CAV*, 334–342.

Cimatti, A.; Giunchiglia, E.; Giunchiglia, F.; and Traverso, P. 1997. Planning via model checking: A decision procedure for ar. In *Recent Advances in AI planning*. Springer. 130–142.

Cimatti, A.; Griggio, A.; Mover, S.; and Tonetta, S. 2015. HyComp: An SMT-based model checker for hybrid systems. In *ETAPS*, 52–67.

Coles, A., and Coles, A. 2013. PDDL+ Planning with Events and Linear Processes. *PCD 2013* 35.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with Problems Requiring Temporal Coordination. In *AAAI*, 892–897.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *ICAPS*, 42–49.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2012. COLIN: Planning with Continuous Linear Numeric Change. *J. Artif. Intell. Res.* 44:1–96.

Della Penna, G.; Magazzeni, D.; Mercorio, F.; and Intrigila, B. 2009. UPMurphi: A Tool for Universal Planning on PDDL+ Problems. In *ICAPS*. AAAI.

Della Penna, G.; Magazzeni, D.; and Mercorio, F. 2012. A Universal Planning System for Hybrid Domains. *Appl. Intell.* 36(4):932–959.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. *IPC at ICAPS*.

Edelkamp, S. 2002. Symbolic pattern databases in heuristic search planning. In *AIPS*, 274–283.

Edelkamp, S. 2014. Planning with pattern databases. In *Sixth European Conference on Planning*.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.* 20:61–124.

Fox, M., and Long, D. 2006. Modelling Mixed Discrete-Continuous Domains for Planning. *J. Artif. Intell. Res.* 27:235–297.

Fox, M.; Long, D.; and Magazzeni, D. 2012. Plan-based Policies for Efficient Multiple Battery Load Management. *J. Artif. Intell. Res.* 44:335–382.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, volume 7, 1007–1012.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *J. Artif. Intell. Res.* 14:253–302.

Hoffmann, J. 2002. Extending FF to Numerical State Variables. In *ECAI*, 571–575. Citeseer.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating"Ignoring Delete Lists"to Numeric State Variables. *J. Artif. Intell. Res.* 20:291–341.

Howey, R., and Long, D. 2003. VAL's progress: The automatic validation tool for PDDL2. 1 used in the international planning competition. In *IPC at ICAPS*.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *ICTAI*, 294–301. IEEE.

Karaman, S.; Walter, M. R.; Perez, A.; Frazzoli, E.; and Teller, S. J. 2011. Anytime motion planning using the RRT. In *IEEE-ICRA*.

Lahijanian, M.; Kavraki, L. E.; and Vardi, M. Y. 2014. A sampling-based strategy planner for nondeterministic hybrid systems. In *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, 3005–3012.

Li, H. X., and Williams, B. C. 2008. Generative Planning for Hybrid Systems Based on Flow Tubes. In *ICAPS*, 206–213.

Maly, M. R.; Lahijanian, M.; Kavraki, L. E.; Kress-Gazit, H.; and Vardi, M. Y. 2013. Iterative temporal motion planning for hybrid systems in partially unknown environments. In *HSCC*, 353–362.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language.

McDermott, D. V. 2003. Reasoning about Autonomous Processes in an Estimated-Regression Planner. In *ICAPS*, 143–152.

Penberthy, J. S., and Weld, D. S. 1994. Temporal Planning with Continuous Change. In *AAAI*, 1010–1015.

Plaku, E.; Kavraki, L. E.; and Vardi, M. Y. 2013. Falsification of LTL safety properties in hybrid systems. *STTT* 15(4):305–320.

Richter, S., and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.* 39(1):127–177.

Shin, J.-A., and Davis, E. 2005. Processes and Continuous Change in a SAT-based Planner. *Artif. Intell.* 166(1-2):194–253.

Tabuada, P.; Pappas, G. J.; and Lima, P. U. 2002. Composing abstractions of hybrid systems. In *HSCC*, 436–450.

Turner, M. J. 2008. *Rocket and spacecraft propulsion: principles, practice and new developments*. Springer Science & Business Media.

Vallati, M.; Magazzeni, D.; Schutter, B. D.; Chrpa, L.; and Mccluskey, T. L. 2016. Efficient Macroscopic Urban Traffic Models for Reducing Congestion: A PDDL+ Planning Approach. In *AAAI*. AAAI Press.

Extended Abstract: Risk-Sensitive Planning with Dynamic Uncertainty

Liana Marinescu
Department of Informatics, King's College London
liana.marinescu@kcl.ac.uk

Publications produced

To cite the heuristic described in this paper, please refer to:

Heuristic Guidance for Forward-Chaining Planning with Numeric Uncertainty (Marinescu and Coles 2016).

1 Introduction

Many compelling applications of planning arise from scenarios that are inherently uncertain. In some cases it is possible to adequately capture the dynamics of the world without modeling uncertainty, and thus to employ classical planning techniques. However, in many other cases it is impossible to ignore uncertainty without hindering the planner's knowledge about the world, and hence obtaining sub-par solutions.

Our research on uncertainty so far is twofold:

- Improving *heuristic guidance* for problems with numeric uncertainty, by including relevant probabilistic information in the heuristic (with a negligible computational cost).
- Extending prior work on *building a policy offline* for problems with nondeterministic action outcomes (Muise, McIlraith, and Beck 2012), by allowing it to support continuous numeric uncertainty.

The first contribution is focused on finding plans for models where there is uncertainty in the outcomes of numeric effects (each governed by a continuous probability distribution). The task is to find a plan where all the preconditions are met, and the goals are reached, with some confidence θ. This paradigm has been explored by previous work, e.g. (Beaudry, Kabanza, and Michaud 2010; Coles 2012); our first contribution is in providing effective *heuristic guidance* in such a setting.

The second contribution – extending prior work on propositional uncertainty to numeric uncertainty – revolves around offline planning. The task is to *build a policy offline* for models where action outcomes have a set of discrete, nondeterministic effects. The class of propositional problems has been addressed by Muise (Muise, McIlraith, and Beck 2012), but numeric uncertainty still remain a challenge. We have extended one of the basic mechanics of policy-building – that of regression (applying an action "backwards" in a state, to obtain a sufficient predecessor state) – from the propositional case, to the case of independently distributed Gaussian numeric uncertainty.

2 Background: Continuous Uncertainty on Actions' Outcomes

To begin with, we build on the state-progression semantics of the planner RTU (Beaudry, Kabanza, and Michaud 2010). Actions have propositional and numeric preconditions and effects, as in classical numeric planning, but the numeric effects have outcomes that are drawn from probability distributions. We say that each effect is of the form $\langle v \ op \ D(\mathbf{v})\rangle$ where $op \in \{+=, =\}$ and D is a (possibly deterministic) probability distribution that governs the range of outcomes of the effect. For example:

- $\langle battery \ \ += \ \ \mathcal{N}(-10, 2^2)\rangle$ – decrease battery by an amount with mean 10 and standard deviation 2.
- $\langle coal \ += \mathcal{N}(15, 3^2)\rangle$ – increase coal by an amount with mean 15 and standard deviation 3.
- $\langle position\_error = \mathcal{N}(0, 1^2)\rangle$ – reset the position error to 0 with standard deviation 1 (e.g. after calibration).

In addition to this, we have a confidence level $\theta \in [0.5, 1)$: because numeric effects have uncertain outcomes, we need to prescribe how certain we must be that each numeric condition is satisfied.

A Bayesian network (BN) is used to define the belief of each $\mathbf{v}$, and as actions are applied, the network is updated with additional variables. In a state S_i, for each $v^j \in \mathbf{v}$, a variable v_i^j is associated with the belief of v. If an action a is applied, leading to a state S_{i+1}, then for each numeric effect $\langle v^j \ \ op \ \ D(\mathbf{v})\rangle$, two random variables are added to the network. The first of these, D_{i+1}^j, represents $D(\mathbf{v})$. The second, v_{i+1}^j, is associated with the belief of v in S_{i+1}, and it is determined by either:

- $v_{i+1}^j = v_i^j + D_{i+1}^j$, if *op* is +=;
- $v_{i+1}^j = D_{i+1}^j$, if *op* is =.

The BN is key to determining whether a plan meets the required confidence level θ. An action a is applicable in a state S_i if $Pre(a)$ is satisfied. A sequential (linear) solution is a sequence of steps $[a_0, .., a_n]$, implying a state trajectory $[I, S_0, .., S_n]$. We use the BN to ensure that with $P \geq \theta$, in a given execution of the plan, each action's preconditions are met and S_n satisfies any hard goals.

The state progression formalism of Beaudry *et al* was adopted and extended by Coles (2012) as the basis of an

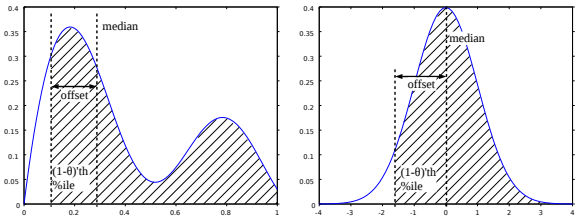


Figure 1: Possible probability distributions: Arbitrary (left) and Gaussian (right).

over-subscription planning approach. A forward-chaining planner following these semantics was used to find a single plan, onto which branches were added by making additional calls to the planner. A range of other approaches have been adopted for planning under uncertainty, such as those based on the use of Markov Decision Processes, e.g. (Meuleau et al. 2009; Mausam and Weld 2008; Rachelson et al. 2008); these approaches are particularly useful when a *policy* needs to be found. As our first contribution is on the heuristic inside a forward-chaining planner, our focus will be on planning under the semantics of RTU described above.

3 Heuristic Guidance for Forward-Chaining Planning with Numeric Uncertainty

3.1 Relaxing Numeric Uncertainty

In deterministic forward-chaining numeric planning, one way to guide search is the Metric Relaxed Planning Graph (RPG) heuristic (Hoffmann 2003). This performs a forward reachability analysis that estimates the number of actions needed to reach goals by relaxing the effects of actions. For numeric state variables, this amounts to estimating reachable bounds on the values of variables, by optimistically assuming that increase effects only increase the upper bound, and decrease effects only decrease the lower bound.

When working with RTU's semantics, Coles (2012) adapted this to assume for heuristic purposes that each variable takes its *median* value. From Jensen's inequality, we know that if $\theta \geq 0.5$, this is guaranteed to be a relaxation. However, as θ becomes large, it also means the heuristic is increasingly unrealistic: a numeric condition might be true assuming variables take their median values; but not when accounting for the uncertainty in their values. In this section, we will present two strategies that improve on this:

- we incorporate the shape of the distribution on variables' values in the heuristic evaluation, rather than discarding it and using the median;
- for Gaussian distributions, we explicitly track the uncertainty of variables in the relaxed planning graph.

Heuristic Guidance with Monotonically Worsening Uncertainty Uncertainty can affect problems in two ways: it either gets worse monotonically (error accumulates and no action can rectify it); or it may be purposefully corrected (there may be actions that reduce the error, such as recharging batteries to a fixed value, or visiting a precise weighing station).

We first discuss the case of monotonically worsening uncertainty. Outside the heuristic, each precondition is of the form $\mathbf{w}.\mathbf{v} \geq c$, and a Monte Carlo simulation is used to estimate the probability distribution of $\mathbf{w}.\mathbf{v}$. Using this distribution, we can test whether the condition is satisfied with probability θ, i.e. whether the $(1-\theta)$'th percentile of $\mathbf{w}.\mathbf{v}$ is $\geq c$. We represent this percentile as follows:

$$p_{1-\theta}(\mathbf{w}.\mathbf{v}) = median(\mathbf{w}.\mathbf{v}) - \mathit{offset}_\theta(\mathbf{w}.\mathbf{v})$$

In effect, offset_θ is the margin of error that must be tolerated, for the precondition to be true with probability θ. We illustrate the intuition behind this margin in Figure 1. The condition itself can then be rewritten:

$$median(\mathbf{w}.\mathbf{v}) \geq c + \mathit{offset}_\theta(\mathbf{w}.\mathbf{v}) \quad (1)$$

We define that uncertainty is *monotonically increasing* if offset_θ *can never decrease*. In this case, it is still a relaxation to use the offset values when determining which preconditions are true in the heuristic – the only way to make the condition true would be to apply actions that affect the values of $\mathbf{v}$, as no actions that decrease offset_θ exist.

An illustrative example would be an autonomous car with a certain amount of fuel, which is used gradually until it runs out; refueling is not possible. The activities performed by the car (e.g. start engine, accelerate, stand still, park) each require fuel, but the amount varies non-deterministically. As the plan is constructed, uncertainty and hence offset_θ accumulates monotonically. We can thus heuristically evaluate a state by assuming offset_θ is constant, and takes its current value; this is guaranteed to be a relaxation, as offset_θ can never become smaller.

Heuristic Guidance with Gaussian Uncertainty So far, we explained how to incorporate distributions on the left-hand side of preconditions ($\mathbf{w}.\mathbf{v}$) into heuristic computation, by using the offset_θ value to capture uncertainty information. The relaxation holds when error accumulates and cannot be lowered. However, problems may contain actions such as *recharge-batteries* or *visit-weigh-station*, which reduce uncertainty.

The challenge in these sorts of problems is to ensure the heuristic remains a relaxation. This is possible in a useful subset of domains, where the uncertainty is due to independent Gaussian-distributed effects on variables, and therefore has an analytic form. We can utilize this form and extend the Metric RPG to additionally track the variance on each variable, $\sigma^2(v)$. The expansion phase, building the RPG, proceeds as follows:

- For each variable $v \in \mathbf{v}$, we track the upper and lower bound on its median value. In the first RPG layer, these are equal to the value of v in the current state S. We additionally track $\sigma^2(v)$, the variance on v. In the first RPG layer, this is the value according to the BN for S.
- In a regular RPG, if a numeric effect is applied that increases (decreases) some $v \in \mathbf{v}$, the upper (resp. lower) bound on v at the next fact layer is updated accordingly. Now, additionally, if a numeric effect decreases $\sigma^2(v)$, the lower bound on $\sigma^2(v)$ at the next fact layer is decreased[1].

[1]Effects increasing $\sigma^2(v)$ are ignored. If $\theta \geq 0.5$, adding more

Algorithm 1: RPG Solution Extraction

Data: RPG, a relaxed planning graph
Result: p, a relaxed plan
1 $last \leftarrow$ last layer index in RPG;
2 $goals[last] \leftarrow G$ *(i.e. the problem goals)*;
3 **for** $l \in [last..0]$ **do for** $(\mathbf{w.v} \geq c) \in goals[l]$ **do**
4 $\quad prev \leftarrow$ max value of $\mathbf{w.v}$ in layer l-1;
5 $\quad prev\_\sigma^2 \leftarrow$ min value of $\sigma^2(\mathbf{w.v})$ in layer l-1;
6 $\quad prev\_offset_\theta \leftarrow prev\_\sigma.\Phi^{-1}(\theta)$;
7 $\quad$ **if** $prev \geq c + prev\_offset_\theta$ **then**
8 $\quad\quad$ add $(\mathbf{w.v} \geq c)$ to $goals[l\text{-}1]$; **continue**;
9 $\quad$ **for** $(w.v) \in \mathbf{w.v}$ *where* $w \neq 0$ **do**
10 $\quad\quad$ Choose actions from l-1 that increase $(w.v)$;
11 $\quad\quad$ Add them to the relaxed plan and subtract their effects from c;
12 $\quad\quad$ **if** $prev \geq c + prev\_offset_\theta$ **then break**;
13 $\quad$ **if** $prev \geq c + prev\_offset_\theta$ **then**
14 $\quad\quad$ add $(\mathbf{w.v} \geq c)$ to $goals[l\text{-}1]$; **continue**;
15 $\quad max\_offset \leftarrow prev - c$;
16 $\quad max\_\sigma^2 \leftarrow (max\_offset/\Phi^{-1}(\theta))^2$;
17 $\quad$ add $(-\sigma^2(\mathbf{w.v}) \geq -max\_\sigma^2)$ to $goals[l]$;
18 $\quad$ add $(\mathbf{w.v} \geq prev)$ to $goals[l\text{-}1]$;

- To decide which actions are applicable in each layer, we take variance into account when checking precondition satisfaction, as follows. For a precondition of the general form $\mathbf{w.v} \geq c$, we can use the additive properties of Gaussians to compute the variance of $\mathbf{w.v}$:

$$\sigma^2(\mathbf{w.v}) = \sum_{w.v \in \mathbf{w.v}} w^2.\sigma^2(v)$$

We obtain the offset using the Gaussian quantile function:

$$offset_\theta(\mathbf{w.v}) = \sigma(\mathbf{w.v}).\Phi^{-1}(\theta)$$

Hence, from Equation 1, the precondition becomes:

$$median(\mathbf{w.v}) \geq c + \sigma(\mathbf{w.v}).\Phi^{-1}(\theta)$$

This gives us everything we need to build an RPG. We can be confident that the $offset_\theta$ values used are relaxations, because smaller values of variance result in smaller values of the Gaussian quantile function Φ^{-1}; and the semantics of the RPG guarantee we will underestimate variance.

The next step is to extract a relaxed plan from the RPG; we illustrate this in Algorithm 1. The first thing to note is on lines 5 and 6, where we compute the $offset_\theta$ necessary for the condition to be met. Actions are then chosen in the standard way to attempt to meet the precondition, given this value of $offset_\theta$. Then, if line 13 is reached and the precondition is still not true, it must mean that a decrease in variance caused it to become true at layer l (having been false at layer l-1). We now need to choose actions that decrease variance enough to achieve this. On line 15, we work out what $offset_\theta$ needs to be reduced to in order to make the precondition true; we then compute its corresponding variance on line 16. This variance can then be used to construct a new condition to be satisfied at this layer: this causes actions to be added to the relaxed plan in order to reduce variance on a later iteration of the loop.

As a result of the algorithm described above, the relaxed plan now contains uncertainty-reducing actions. This makes for a better-informed heuristic, which is able to provide improved guidance and dead-end detection to the search, as will be demonstrated in the following section.

uncertainty never contributes towards preconditions becoming true, so it suffices to track only the smallest reachable values of variance.

3.2 Evaluating the new Heuristic

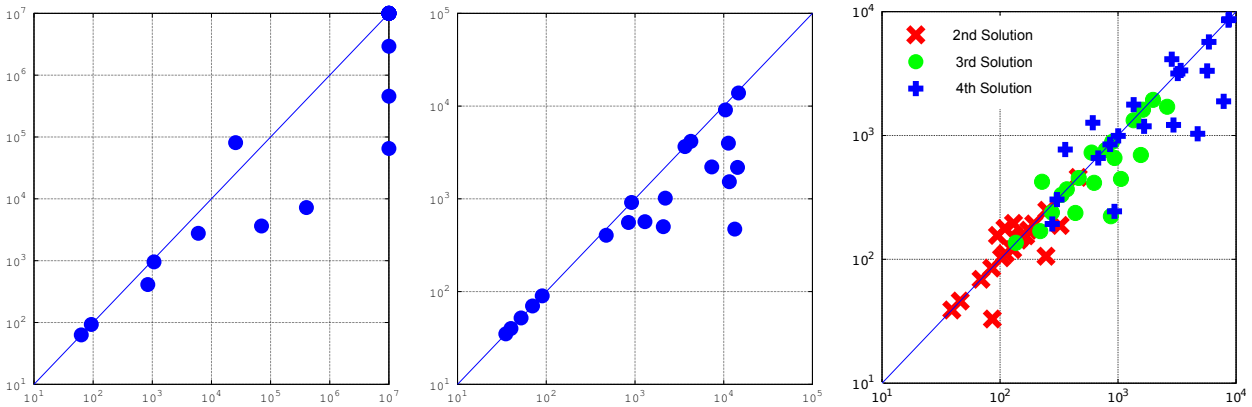
We evaluate on three domains: Rovers and AUV from (Coles 2012); and a variant of TPP from (Gerevini et al. 2009). In Rovers, the activities of a planetary rover are constrained by battery usage, which has Gaussian uncertainty, and the battery can only be recharged at certain locations. In TPP, the domain is modified to model the acquisition of sufficient amounts of bulk materials (e.g. coal), and trucks can visit weighing stations at some suppliers to top up or shed excess load, which reduces uncertainty. AUV is an over-subscription problem where the activities of an underwater vehicle must be planned with a strict bound on total time taken, and with normally distributed activity durations. Tests were performed on 3.5GHz Core i5 machines with a limit of 4GB of memory and 1800s of CPU time.

Overall, the new heuristic leads to faster planner performance; the time-to-solve scatterplots look the same as the nodes-generated scatterplots in Figure 2. The extra computational work (tracking variances etc.) does not adversely affect the time taken to heuristically evaluate a state. Thus, because significantly fewer states need to be evaluated, and state evaluation times are comparable, the performance of the planner is significantly better.

For the Rovers domain (Figure 2a), most striking are the points on the far right of the graph – these indicate problems that were previously unsolvable but can now be solved. In part, this is because the new heuristic is able to recognize many more states as being dead ends, because it does not disregard uncertainty on the battery level when evaluating preconditions. In contrast, by ignoring uncertainty, the old relaxed plans relied on moving somewhere to recharge, even though in reality uncertainty made it impossible for that move action to be applied. The new heuristic often avoids this pitfall by accounting for uncertainty to a greater extent.

In TPP (Figure 2b), all the problems could be solved by both the old and the new heuristic. However, by not accounting for uncertainty, the old heuristic can reach states in which the relaxed plan does not need to buy any more of any goods. In these states, the heuristic value is 0. As acquiring additional goods requires combinations of travel and buy actions, a substantial amount of search must be performed with no effective heuristic guidance. Unlike Rovers, there are no dead ends due to these travel actions, so this blind search will succeed, but is very time consuming – in problems furthest from the line $y = x$, the majority of nodes evaluated have an old heuristic value of 0.

AUV is an over-subscription problem: search reports a solution plan every time it finds one that solves more goals than the best so far. We are hence interested in the search



(a) Rovers (b) TPP (c) AUV

Figure 2: Nodes generated to solve problems in the three evaluation domains. Axes are logarithmic, comparing prior work (X axis) with the new heuristic (Y axis). The Two-Tailed Wilcoxon Signed-Rank Test confirms results are significant to $P \geq 0.95$.

effort to find progressively better solutions. Figure 2c compares the nodes generated by each configuration to find the 2nd, 3rd and 4th solutions. (These correspond to satisfying 1, 2, and 3 goals respectively.) The relaxed plans produced by the old heuristic, by ignoring uncertainty, more often use actions that there is actually no time to complete. Disregarding uncertainty is less of an impediment than in Rovers and TPP, as there is no scope for planning actions that reduce uncertainty (unlike battery charge or goods purchased, actions cannot create more time). Nonetheless, the new heuristic is generally able to find better solutions more quickly. If left to run for long enough, search with the old heuristic will tend to find solutions as good as search with the new heuristic, but loses out earlier in the search.

As a concluding remark for our results, we note that so far we assumed $\theta = 0.99$. At $\theta = 0.8$, the improvements from using the new heuristic are still noticeable, but not as substantial. By $\theta = 0.6$, which is close to the median ($\theta = 0.5$), there is no statistically significant difference between the two, as uncertainty has only a modest effect on the heuristic, or indeed search itself. This confirms that our heuristic meets our headline aim of being able to better guide the planner when the consequences of uncertainty bear a significant effect upon what is a reasonable solution plan.

4 Background: Discrete Uncertainty on Actions' Outcomes

The next step in our research on uncertainty revolves around problems with discrete non-deterministic action outcomes. In order to reason about discrete non-determinism, we use actions that have a precondition (as in the deterministic case), but instead of having a single effects list, they have several. Applying an action will nondeterministically trigger one of these. We assume states are *fully observable* – we can observe what effects an action had. A solution to problems containing such actions can be represented by using a policy – a set of rules that dictates what should be done in each state.

Since action outcomes are discretely different, a policy may need to branch out and apply different actions in the different states reached. A weak plan corresponds to a single trajectory of actions that leads from the initial state to a goal state, assuming it is possible to choose which action outcome occurs at each point. Weak plans can be found using a deterministic planner given as input the *all outcomes determinisation* (Yoon, Fern, and Givan 2007) – the representation of an action with discrete outcomes as a set of actions having identical preconditions but different lists of effects (one list corresponding to each discrete outcome).

Muise *et al.* (2012) present an approach where, by repeatedly invoking a deterministic planner to find weak plans, it is possible to incrementally build a policy. Key to the success of their approach is exploiting *relevance* – by regressing the goal through a weak plan step-by-step, they determine which facts at each point are relevant to plan success.

Regression begins from the goals, which here are a set of propositions, ps'. Regressing ps' through an action A, with preconditions $pre(A)$ and add effects $add(A)$ yields a partial state ps where:

$$ps = (ps' \setminus add(A)) \cup pre(A)$$

A policy can then be built from these pairs, each $\langle ps, A \rangle$. If executing the policy reaches a state where $S \vDash ps$, then A is applied. Otherwise, if there is no such match, the planner is invoked from S, producing another weak plan which is added to the policy in this way. Policy building terminates when $\exists \langle ps, A \rangle . S \vDash ps$ for all states S reachable from the initial state, via the action choices indicated by the policy.

One caveat of this process is that it must be restarted if a dead-end state D is reached. If this happens, Muise *et al.* record *forbidden state–action pairs*: D is regressed through all actions $[a_0..a_n]$ that could lead to it, yielding states $[S_0..S_n]$; then each $\langle S_0, a_0 \rangle$ is forbidden, as applying a_i in S_i would reach D once again. To improve generality, each state S_i is *generalised* – a greedy algorithm is used to remove facts that do not affect whether S_i is a dead-end.

The policy produced by this process, overall, is a strong cyclic plan: it will always reach the goals, if the goals can be reached. The work has since been developed to support conditional effects (Muise, McIlraith, and Belle 2014) and sensing actions (Muise, Belle, and McIlraith 2014), which

are interesting avenues we hope to look at in the future. For now, we concern ourselves with extending this well-defined propositional formalism to incorporate continuous numeric uncertainty.

5 Building Policies Offline for Continuous and Discrete Uncertainty

5.1 Allowing Numeric Uncertainty in Action Effects

As detailed in Section 3, we have a planner kernel that is capable of supporting actions with continuous numeric uncertainty. Additionally, as detailed in Section 4, Muise *et al.*'s work on offline policy-building currently supports propositional effects. It is therefore a natural step to use our planner to extend their work to numeric effects, thus allowing a richer set of problems to be tackled.

Furthermore, as we will describe below, extending previous work in this manner provides more than just the ability to build policies with numeric effects. It retains and generalises the mechanics we defined previously for dealing with numeric uncertainty. This greatly increases the level of realism that problems can reflect, as continuous uncertainty (e.g. will we step 10 metres or 10.5 metres?) can now function alongside discrete uncertainty (e.g. will we step forward or will we blow a fuse?).

5.2 Representing Partial States with Numeric Constraints

The approach of Muise *et al.* targets problems with propositional constraints. In their case, partial states can be intuitively defined as containing only a subset of propositional constraints (e.g. the full state {at-home, have-package, is-birthday} satisfies the partial state {at-home}). We extend this intuition to numeric constraints: if a full state contains the constraints $\{battery{=}15, altitude{=}40\}$, then this can satisfy a partial state $\{battery \geq 10\}$. But, we must additionally take care to account for any variance on the value of $battery$ at this point.

We first define a representation of a constraint that includes an explicit record of any variance that needs to apply to it. For a constraint $\mathbf{w}.\mathbf{v} \geq c$ we record a constraint tuple $\langle ft, c, vt, av \rangle$ where:

- ft are the formula terms, initially the weighted-sum of variables, $\mathbf{w}.\mathbf{v}$.
- c is the right-hand-side constant, initially c.
- vt are variance terms. These are initially $\{\langle w_0, \sigma^2(v_0), 0\rangle..\langle w_n, \sigma^2(v_n), 0\rangle\}$ for each $w_i.v_i \in \mathbf{w}.\mathbf{v}$.
- av is an accumulated variance value, initially 0.

Performing regression is akin to applying an action "backwards" in a state, to obtain the sufficient conditions for that action application to have resulted in that state. If a constraint C is regressed through a numeric effect v+=$\mathbf{w}'.\mathbf{v}'+e$, where $w.v \in ft$, the resulting constraint C' is:

$$\begin{aligned} ft' &= ft + w.(\mathbf{w}'.\mathbf{v}') \\ c' &= c - w.e \\ vt' &= vt \\ av' &= av \end{aligned}$$

Regressing through the effect $v = \mathbf{w}'.\mathbf{v}' + e$ gives:

$$\begin{aligned} ft' &= ft - w.v + w.(\mathbf{w}'.\mathbf{v}') \\ c' &= c - w.e \\ vt' &= vt \\ av' &= av \end{aligned}$$

Effects on variance also affect the variance portions of constraints. If $\sigma^2(v)$ is changed by an effect $\sigma^2(v)$+=e then for any constraint with an element $\langle w, \sigma^2(v), k\rangle \in vt$, vt' is identical modulo k being increased by e. For the effect $\sigma^2(v) = e$, then for any constraint with an element $\langle w, \sigma^2(v), k\rangle \in vt$, vt' is identical modulo this element being removed, and $av' = av + e.w^2$. Conceptually, after regressing through this effect assigning variance a fixed value, any earlier effects on variance are moot: it suffices to transfer the newly assigned (and weighted) amount to the accumulated variance.

To determine if a state S satisfies a constraint C with confidence θ, we define the Gaussian distribution with mean ft (taking values of variables from S), and with variance:

$$av + \sum_{\langle w, \sigma^2(v), k\rangle \in vt} w^2.(S[\sigma^2(v)] + k)$$

Then, if the $1-\theta$'th percentile of this Gaussian is $\geq c$, we say S satisfies C. Effectively, the condition must be true allowing for the variance in S, and any in the condition itself. This is a slight departure from checking preconditions on actions as described in Sections 2 and 3, where the only variance to account for was that in S. Here, the constraints denote preconditions that must be true *later in the weak plan*, so the variance within the constraint tracks the impact of effects on variance between S and when the precondition must in fact hold.

5.3 Handling Dead Ends

Muise *et al.* introduced a particularly insightful contribution concerning dead ends. As mentioned in Section 4, their idea is to mark partial state–action pairs as forbidden if they lead to a dead end. Whenever a new dead end is found while building the policy, its corresponding partial state–action pair (i.e. the pair that would lead to that dead end) is generated via one step of regression, and stored in the forbidden list. The policy is then deleted, and policy building restarted with this new information at hand.

We build our extension along the same lines. As we are performing numeric planning, our states are split into two parts: logical and numeric. The former can be handled exactly as by Muise *et al.*: regressed through the logical effects of the action. For the latter, for each variable value v=k in the state, following the representation set out in Section 5.2 we build pairs of tuples $\langle v, k, \{\}, 0\rangle$, $\langle -v, -k, \{\}, 0\rangle$. (Note this equally applies to variance-tracking variables – which are first-class citizens and appear as state variables along with the others.) These numeric constraints are then regressed through the numeric effects of the action.

To generalise these dead-ends, we again look at the logical and numeric parts of the state. For the logical, the approach of Muise *et al.* based on the RPG heuristic can be used: in Section 3, we fortuitously described exactly the sort

of RPG heuristic that would be needed to support this. For the numeric case, if static analysis reveals that larger/smaller values of a variable are better in terms of satisfying conditions, then we need only keep one constraint tuple, not a pair. Simply, for a dead-end state where v=k, if we know larger values of v are better, we only need record $\langle -v, -k, \{\}, 0\rangle$ (which is analogous to $v \leq k$), as any state with this value of v or worse is going to be a dead end.

5.4 Preliminary Observations

So far, we have conducted some preliminary tests on a modified version of the rovers domain, extended to better model the distribution of energy usage when moving. Rather than assuming this can be adequately captured by a single Gaussian-distributed variable, there is a pessimistic outcome with high energy usage corresponding to a failure mode of the rover; and an optimistic outcome corresponding to an unusually clear path. This leads to a policy being built that considers what to do for each of these outcomes.

It is clear at this point that the power of partial states seen in the propositional case, is also being seen in the numeric case. If a pessimistic navigate outcome occurs, then the policy does not need to branch if there is still enough power left; i.e. the resulting state still satisfies the relevant partial state. Even if branching does occur (i.e. the planner needed to be invoked to find another weak plan) the recharge actions have an interesting effect. As these recharge the battery and clear the variance on its value, there are no restrictions on battery charge in the partial state before them. Thus, even if the policy splits into distinct branches, these often later merge.

Generalising dead-ends for numeric resources also has a beneficial effect: in plain English, the dead-ends seen can generally be interpreted as 'if the rover is in this location, with only this much battery charge, or too much variance on battery charge, then applying this action will lead to a dead-end'. These situations arise when the rover would be unable to reach somewhere it can recharge and then resume operations. When restarting policy building, the dead-ends found are effective in pruning unsuitable action choices from search.

6 Future work

So far we have successfully improved heuristic guidance in problems with numeric uncertainty. We are also well into providing an extension to propositional offline policy-building, allowing it to work in problems with numeric action effects. The next stage consists of finalising this extension, and conducting a detailed evaluation.

We will also allow our planner to accept domains where there is non-Gaussian uncertainty. For this, we would generalise the techniques we already implemented and tested, in order to make the transition from Gaussian probability distributions to arbitrary probability distributions. This would considerably broaden the spectrum of problems the planner will be able to solve. In particular, we intend to incorporate Bayesian Network techniques to support non-Gaussian uncertainty, and adapt regression and probabilistic dead ends for this type of uncertainty.

The next promising avenue to explore afterwards would be to apply our technique to a "planning in the loop" setting, with the scope for adjusting the probability distributions of action outcomes based on experience gathered from plan execution so far. As a practical case study, we plan to test our approach on an autonomous robotic platform operating in a dynamic environment. A concrete example would be a quadcopter conducting a rescue mission inside a building that is still collapsing, or seeking out an outdoor waypoint amongst vegetation or foliage. The success of practical testing would indicate the programme of work has had its desired outcomes: developing planning approaches that function well with more realistic world dynamics, and broadening the spectrum of possible applications of planning.

References

Beaudry, E.; Kabanza, F.; and Michaud, F. 2010. Planning with Concurrency under Resources and Time Uncertainty. In *Proceedings of ECAI*.

Coles, A. J. 2012. Opportunistic Branched Plans to Maximise Utility in the Presence of Resource Uncertainty. In *Proceedings of ECAI*.

Gerevini, A.; Long, D.; Haslum, P.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic Planning in the Fifth International Planning Competition: PDDL3 and Experimental Evaluation of the Planners. *Artificial Intelligence*.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating Ignoring Delete Lists to Numeric State Variables. *Journal of Artificial Intelligence Research* 20.

Marinescu, L., and Coles, A. 2016. Heuristic guidance for forward-chaining planning with numeric uncertainty. In *Proceedings of ICAPS*.

Mausam, and Weld, D. S. 2008. Planning with Durative Actions in Stochastic Domains. *Journal of Artificial Intelligence Research* 31.

Meuleau, N.; Benazera, E.; Brafman, R. I.; Hansen, E. A.; and Mausam. 2009. A Heuristic Search Approach to Planning with Continuous Resources in Stochastic Domains. *Journal of Artificial Intelligence Research* 34.

Muise, C.; Belle, V.; and McIlraith, S. 2014. Computing contingent plans via fully observable non-deterministic planning. In *Proceedings of AAAI*.

Muise, C.; McIlraith, S.; and Beck, J. 2012. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of ICAPS*.

Muise, C.; McIlraith, S.; and Belle, V. 2014. Non-deterministic planning with conditional effects. In *Proceedings of ICAPS*.

Rachelson, E.; Quesnel, G.; Garcia, F.; and Fabiani, P. 2008. A Simulation-Based Approach for Solving Temporal Markov Problems. In *Proceedings of ECAI*.

Yoon, S.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *Proceedings of ICAPS*.

Session 2

Multi Agent Planning & Plan Execution

A Distributed Online Multi-Agent Planning System (Dissertation Abstract)

Rafael C. Cardoso
{rafael.caue@acad.pucrs.br}

Supervisor: Rafael H. Bordini
FACIN-PUCRS
Porto Alegre - RS, Brazil

Abstract

The gap between planning and execution is still an open problem that, despite several tries from members of both automated planning and autonomous agents communities, remains without a proper general-purpose solution. We aim to tackle this problem by using a framework for the development of multi-agent systems in both the decentralised multi-agent planning stages, and the execution stages, providing a multi-agent system with capabilities to solve online multi-agent planning problems.

1 Introduction

Multi-Agent Systems (MAS) are often situated in dynamic environments where new plans of actions need to be constantly devised in order to successfully achieve the system goals. Therefore, employing planning techniques during run-time of a MAS can be used to improve agent's plans using knowledge that was not previously available, or even to create new plans to achieve some goal for which there was no known course of action at design time.

Research on automated planning has been largely focused on single-agent planning over the years. Although it is possible to adapt centralised single-agent techniques to work in a decentralised way, such as in (Crosby, Jonsson, and Rovatsos 2014), distributed computation is not the only advantage of using Multi-Agent Planning (MAP). By allowing agents to do their own individual planning the search space is effectively pruned, which can potentially decrease planning time on domains that are naturally distributed. This natural distribution also means that agents get to keep some (or even full) privacy from other agents in the system, as they might have beliefs, goals, and plans that they do not want to share with other agents. Single-agent planning can have no privacy, since the planner needs all the information available.

MAS went through a similar process of transitioning from single to multiple agents, albeit at a faster rate. Recent research, as evidenced in (Boissier et al. 2011; Singh and Chopra 2010), shows that considering other programming dimensions such as environments and organisations as first-class entities along with agents allow developers to create more robust MAS.

Thus, in this dissertation abstract we introduce the design of our Distributed Online Multi-Agent Planning System (DOMAPS). DOMAPS is composed of: i) a formalism for the representation of decentralised domains and problems in multi-agent planning, based on Hierarchical Task Network (MA-HTN); ii) a contract net protocol mechanism for goal allocation; iii) individual planning with the SHOP2 planner; and iv) the use of social laws to coordinate the agents during execution. Some preliminary results from initial experiments in a novel scenario, the floods domain, are shown.

Although approaches to online single-agent planning usually involve some kind of interleaving planning and execution (e.g., lookahead planning), in our initial approach to online multi-agent planning we focus on domains that allow agents some time to plan while the system is still in execution. DOMAPS allows for the dynamic execution of plans found during run-time, making it easy to transition from planning into execution and vice-versa, while still permitting agents to continue their execution, as long as their actions are believed to not cause any conflict with actions from a possible solution.

The remainder of the dissertation abstract is structured as follows. In the next section a discussion on multi-agent planning is presented. Section 3 introduces the initial design of the Distributed Online Multi-Agent Planning System (DOMAPS). Next, in Section 4, we describe the implementation of DOMAPS in a MAS development framework. In Section 5, we describe the floods domain, a novel domain designed for heterogeneous multi-agent systems. Some initial experiments using DOMAPS in this domain are also shown. We conclude with a discussion on related work and some concluding remarks.

2 Multi-Agent Planning

Multi-Agent Planning (MAP) has often been interpreted as two different things. Either the planning process is centralised and produces distributed plans that can be acted upon by multiple agents, or the planning process itself is multi-agent. Recently, the planning community has been favouring the concept that MAP is actually both of these things, that is, the planning process is done **by** multiple agents, and the solution is **for** multiple agents.

When considering multiple agents, the planning process gets increasingly more complicated, giving rise to several problems (Durfee and Zilberstein 2013). Actions that agents choose to make may cause an impact in future actions that

the other agents could take. Likewise, if an agent knows which actions the other agents plan to take, it could change its own current choices. When dealing with multiple agents, concurrent actions are also a possibility that may require additional care.

In Table 1 we characterise some differences between single-agent planning and multi-agent planning. Although computation can be distributed in single-agent planning it is not commonplace, since the cases where it is actually useful are too few we omitted it from the table. And while multi-agent planning could have no privacy, even in fully cooperative domains it is fairly trivial to allow at least some sort of partial privacy. Full privacy on the other hand is quite difficult because of the coordination needs in multi-agent planning. Single-agent planning has no privacy, since the planner needs all the information available. Agents in single-agent problem formalisms are usually represented as any other object or fact of the environment. Agents in multi-agent planning are treated as first-class entities, where each agent can have its own domain and problem specification.

Table 1: Comparisons between single-agent planning (SAP) and multi-agent planning (MAP).

	computation	**privacy**	**agent abstraction**
SAP	centralised	none	objects
MAP	decentralised	partial or full	first-class entities

Durfee, in (Durfee 1999), establishes some stages of multi-agent planning, that were further extended in (Weerdt, Mors, and Witteveen 2005) and (de Weerdt and Clement 2009):

1. **Global goal refinement:** decomposition of the global goal into subgoals.
2. **Task allocation:** use of task-sharing protocols to allocate tasks (goals).
3. **Coordination before planning:** coordination mechanisms that prevent conflicts during the individual planning stage.
4. **Individual planning:** planning algorithms that search for solutions.
5. **Coordination after planning:** coordination mechanisms that correct conflicts during the individual planning stage.
6. **Plan execution:** the agents that participated in the planning process now carry out the plans.

Not all of these stages are necessary in MAP, some may even be combined into one.

3 The Distributed Online Multi-Agent Planning System

Our multi-agent planning system consists of several main components: **planning formalism** – formally describes the information from the planning domain and problem that will be used during planning; **goal allocation** – set of techniques used to allocate goals to agents; **individual planning** – the planner used during each agent's individual planning stage; and **coordination mechanism** – used before or after planning to avoid possible conflicts that can be generated during planning. DOMAPS was made to work as a general-purpose domain-independent system, and as such we expect to turn it into an open platform where many other alternatives for main components can be added, allowing the MAS designer to pick and choose the ones that work better for their particular distributed online multi-agent planning problem.

Currently, DOMAPS can be used in three different situations where agents have access to the following commands:

- **plan:** plan for a set of organisational goals in which there are no know plans.
- **replan:** plan for a specific organisational goal, either because the known plan failed, or because the agent detected a change in the environment that could potentially lead to a better solution.
- **replanall:** drop all current organisational goals and their related intentions, and start a new planning process for the organisational goals that were dropped, using up-to-date information about the environment. Useful in case everything seems to be going wrong, though this construct is slightly more difficult to automate than the other two.

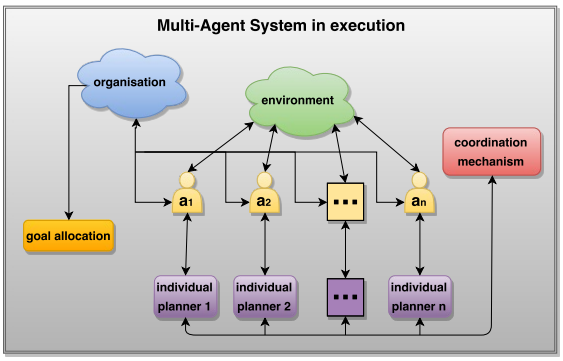


Figure 1: DOMAPS design overview.

The design overview of DOMAPS is shown in Figure 1. Multiple agents $(a_1, a_2, ..., a_n)$ interact with an environment to obtain information and carry out their actions. These agents are part of an organisation, adopting roles and following norms and receiving missions that are related to their roles, all the while pursuing the organisation's goals.

The planning process in DOMAPS consists of the following: a mechanism is used to separate and allocate goals to agents; up-to-date information is collected and translated into a planning formalism that the planner can understand; agents start their individual planner in the search for a solution to the set of allocated goals; agents coordinate with each other either before or after the planning process, in order to prevent the generation of any conflicts or help solve any dependencies; and finally, each agent translates the solution found by their respective planners into plans that can be added to their plan library.

3.1 Planning Formalism

We propose the Multi-Agent Hierarchical Task Network (MA-HTN) formalism, which is an extension of the centralised single-agent HTN formalism used in the SHOP2 planner (Nau et al. 2003). MA-HTN is intended for **online** multi-agent planning problems, since domain and problem information have to be collected during execution. Agents use a **translator** to parse their information about the world into domain and problem specifications that is then passed to their own individual planner.

Each agent has their own problem and domain specification. This provides a decent level of privacy on its own, since each planner only has access to their respective agent problem and domain specifications. This means that, unlike some of the other multi-agent planning formalisms, MA-HTN does not need to have privacy or public blocks. Although at some point it might be interesting to add the capability to include private goals into the planning consideration, for now we are interested only on organisational goals.

Actions from other agents can cause conflicts, either at the moment that action is executed (e.g., concurrent actions) or in the future (e.g., durative actions). Actions that can cause **conflict** have to be annotated by the MAS developer, in order for the translator to identify them. Likewise, dependencies between actions can also exist, either as a concurrent action that requires another agent or as actions that depend on the actions of other agents to happen first. These **dependency** relations also have to be annotated by the MAS developer, so that the translator can add them to the specification.

3.2 Goal Allocation

A Contract Net Protocol (CNP) mechanism is used to allocate goals to agents in DOMAPS. Our CNP mechanism is based on the original CNP design of Reid G. Smith (Smith 1980), with a few modifications in order to accommodate our needs for a goal allocation mechanism in the context of MAP. The initiator in our case will always be the organisation. It is the organisation's role to start new auctions for organisational goals that do not have any known plans on how to achieve them, or for organisational goals that have plans, but needs to be re-planned. The bidders, then, are the agents that are part of the organisation and participate during planning.

The logic of the bid depends on the rest of the mechanisms being used in DOMAPS and in the MAS development platform, but it is fair to assume that agents have the ability of checking their plan library for plans that are able to decompose, at least at some level, the goal that is being auctioned. Although domain-dependent procedures for determining the bid will provide better results, we provide a simple domain-independent general-purpose procedure that agents can use to determine their bid, shown in Algorithm 1.

The agent checks if the announcement of the goal came from the organisation and if he is eligible according to the eligibility criteria provided in the announcement, or otherwise decides not to bid. If the agent chooses to proceed with the bid, then, he keeps decomposing the goal into subtasks and incrementing the bid by 1 for each level that was successfully decomposed, either until it is close to the deadline, or it arrived in an action that could achieve the goal, or it found a dead end (in which case the bet is null).

Algorithm 1 Domain-independent algorithm for determining an agent's bid.

```
procedure bid (from, goal-name, goal-spec, eligibility, deadline)
if from ≠ organisation then return failure
else if I am not eligible then return failure
else
    while ((close to deadline) or (no more levels available to decompose)) do
        decompose one level of one task from goal-spec
        bid-value = bid-value++
    end while
return bid-value
end if
```

By the end of the loop the bidder agent will have the value of the bid to be sent to the initiator. The initiator allocates the goal to the agent with the lowest (not null) bid. We assume here that every goal will eventually be allocated, meaning that there is at least one agent eligible for each organisational goal.

3.3 Individual Planner

SHOP2 (Nau et al. 2003) is an HTN planner with support for the sort of anytime planning that DOMAPS requires. No modifications were made to the actual planning algorithm and search techniques of SHOP2, as the multi-agent mechanisms present in the other components proved to be enough for our initial experiments. As long as we can keep the individual planners intact, DOMAPS benefits from its multi-layered approach, making it easier to change components as we see fit, with little to no modification required in the planners.

Many parameters can be used to tweak the SHOP2 planner. Perhaps the most relevant to DOMAPS is the parameter that guides which kind of search that will be made, of which the possible values are:

- **first:** depth-first search that stops at the first plan found.
- **shallowest:** depth-first search for the shallowest plan, or the first such plan if there are more than one.
- **id-first:** iterative-deepening search that stops at the first plan found.

3.4 Coordination Mechanism

Social laws can coordinate agents by placing restrictions on the activities of the agents within the system. The purpose of these restrictions is twofold: it can be used to prevent some destructive interaction from taking place; or it can be used to facilitate some constructive interaction.

The design of social laws is domain-dependent, and we require them to be supplied by a designer offline. Thus, while the social laws are provided before planning, we do not directly use them during individual planning. Instead, we take advantage of the capabilities provided by the

MAS development framework, that we used to implement DOMAPS, in order to apply social laws in coordination after planning.

In the original model of Shoham and Tennenholtz (Shoham and Tennenholtz 1995), social laws were used to restrict the activities of agents so as to ensure that all individual agents are able to accomplish their personal goals. We follow a similar idea, although agents here aim to achieve organisational goals, and thus, are naturally compelled to follow the social laws that are present in the system.

We formally define social laws in our model as:

Definition 1 Given a set of agents Ag, a set of actions Ac, a set of states S, a set of preconditions P, and a set of options Θ, a *social law* is a tuple (ag,ac,s,P,Θ) where $ag \in Ag$, $ac \in Ac$, and $s \in S$.

A social law sl constrains a specific action ac of agent ag, considered to be a possible point of conflict (as established in the operator description, as shown in the MA-HTN formalism), when the state s satisfies each precondition $\rho_i \in P$, giving the agent all possible options $\theta_i \in \Theta$. Although not explicitly present in this model, the *null* action (e.g., do nothing) can be a possible option, but in order for it to be viable it needs to have been established as an action (operator) in the MAS.

4 Multi-Agent System Integration

DOMAPS is an online system, and, as mentioned before, that implicates the use of planning techniques whilst the MAS is running. Therefore, we need a MAS development platform in order to properly implement and evaluate DOMAPS. We chose to use the **JaCaMo**[1] (Boissier et al. 2011) framework as the MAS development platform, since it contains all of the programming abstractions that DOMAPS requires – **organisation**, **environment**, and **agent** abstractions.

JaCaMo combines three separate technologies into a framework for MAS programming that makes use of multiple levels of abstractions, enabling the development of robust MAS. Each technology (Jason, CArtAgO, and Moise) was developed separately for a number of years and are fairly established on their own when dealing with their respective abstraction level (agent, environment, and organisation).

To illustrate the run-time of DOMAPS when the `domaps.plan` internal action is executed, consider the overview provided in Figure 2. When an agent executes `domaps.plan`, it goes through phase 1 and activates the contract net protocol to allocate the organisational goals between the agents. Then, in phase 2, each agent knowledge about the world is passed to a MA-HTN translator, that sends the information needed to SHOP2 for the individual planning that takes place in phase 3. The solution found by each agent's planner goes back through the MA-HTN translator again, translating the solution into AgentSpeak plans. Finally, the solution is carried out by the agents in accordance to the social laws (phase 5) that are associated with actions from the solution that can cause conflicts.

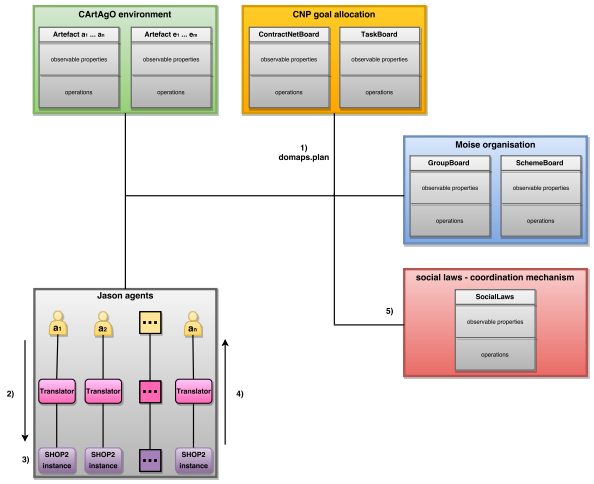


Figure 2: DOMAPS run-time overview of the `domaps.plan` internal action.

5 The Floods Domain

The lack of robust and complex multi-agent domains led us to design a new domain, in order to best exploit the advantages of MAP and MAS. The inspiration for this specific domain came from a real-world scenario, taken from another project that we currently participate. It is a multidisciplinary and inter-institutional project that focuses on using information technology (e.g., a team of autonomous multi-robots) to help mitigate and prevent natural disasters. This scenario is specifically targeted at flood disasters, often caused by intense hydro-meteorological hazards that can lead to severe economic losses, and in some extreme cases even deaths.

Our domain, the Floods domain, is based on that real-world scenario. In the floods domain, a team of autonomous and heterogeneous robots are dispatched to monitor flood activity in a region with multiple areas that are passive of floods. All of the goals come from the Centre for Disaster Management (CDM) that is located in the region being monitored. The CDM is usually operated by humans, but in our JaCaMo+DOMAPS implementation we simulate them by using agents, capable of creating dynamic goals during run-time.

In Figure 3, we show the elements that compose the Floods domain. The domain takes place in a particular region, which is divided into several interconnected *areas*. Movement through the region occurs from traversing these areas. *Flood* events are common in the region, especially during heavy-rain. These floods can be observed from specific areas in the region. The areas can be connected by a *water path*, that can be traversed by naval units, and/or by a *ground path*, that can be traversed by ground units. *Water*

[1] `http://jacamo.sourceforge.net/`.

sample can be requested to be collected from certain areas. During flood events, *victims* may be detected and in need of assistance. The *CDM* establishes a base of operations in one of the areas in the region.

Finally, the naval units are composed of *USVs* that can move through areas connected by water paths, collect water samples, and take pictures of flood events. Meanwhile, the *UGVs* are ground units that are able to move through areas connected by ground paths, take pictures of flood events, and provide assistance to victims by transporting first-aid kits to first responders close by. The robots can only perceive other robots that are in the same area.

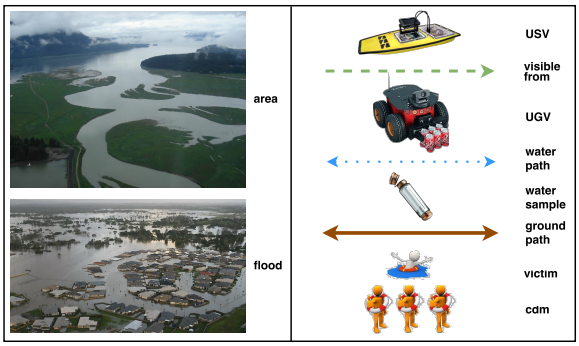


Figure 3: Elements from the Floods domain.

5.1 Experiments

For these initial experiments, we maintained the number of agents and focused on increasing the number of goals. It seems that there is a relation between the number of goals and the number of agents. For most domains, having the number of goals equal to the number of agents, and assuming that each agent is capable of solving its associated goal, appears to result in faster planning times. As the number of goals surpasses the number of agents, the planning time approximates to that of single-agent SHOP2. In Table 2 we show the some initial experiments on this domain for small problems with 4, 8, 16, and 32 goals.

The results are shown in regards to time spent planning, and the number of state expansions and inferences that were made during planning. These results do not depict any of the run-time features of DOMAPS, as we are still investigating how to evaluate it as a whole, and considering what evaluation parameters that could be used both for planning and for execution.

It is clear that our approach would be faster than regular SHOP2, since we are assigning goals to agents previously, while SHOP2 needs to expand states during planning in order to try different assignments. The real advantage that these initial experiments show relate to the number of expansions and inferences, showing DOMAPS does much fewer, even if adding all the agents, than SHOP2. The individual planning approach taken in DOMAPS can discard many of the predicates that are usually used to assign tasks between different objects, remember that agents in SHOP2 are no different than any other object from the planning formalism. By

Table 2: Initial experiment results.

	DOMAPS			**SHOP2**
	usv1	usv2	ugv1	
floods 4 pl. time	0.001	0.001	0.001	0.004
exp.	8	8	15	65
inf.	13	13	21	186
floods 8 pl. time	0.001	0.001	0.002	0.011
exp.	15	15	29	129
inf.	21	21	37	360
floods 16 pl. time	0.002	0.002	0.004	0.033
exp.	29	29	57	257
inf.	37	37	69	708
floods 32 pl. time	0.003	0.003	0.005	0.095
exp.	57	57	113	513
inf.	69	69	133	1404

using agents as first-class abstractions during planning we are free of the use of these predicates. These results should also be scalable, which we aim to prove in future experiments. Experiments for increasing the number of agents, and also for increasing the number of predicates, are already underway.

6 Related Work

There has been several surveys over the years describing advancements in particular areas of planning. Of interest and related to this research there are, for example: in (desJardins et al. 1999), a survey on distributed online (continual) planning is presented, with the state of the art in distributed and online planning at the time (1999), and a design for a distributed online planning paradigm; a survey (Meneguzzi and De Silva 2013) that presents a collection of recent techniques (2013) used to integrate single-agent planning in BDI-based agent-oriented programming languages, focusing mostly on efforts to generate new plans at run-time; and two multi-agent planning surveys, in 2005 (Weerdt, Mors, and Witteveen 2005) and 2009 (de Weerdt and Clement 2009), describing several approaches taken towards multi-agent planning over the last few years.

In (Nissim and Brafman 2014), the authors propose a heuristic forward search for classical multi-agent planning that respects the natural distributed structure of the system, preserving agent privacy. According to their experiments, their system showed the best performance in regards to planning time and communication, as well as the quality of the solution in most cases, when compared to other offline multi-agent planning systems.

FLAP (Sapena, Onaindia, and Torreño 2015) is a hybrid planner that combines partial-order plans with forward search and uses state-based heuristics. FLAP implements a parallel search technique that diversifies the search. Unlike the other planners, FLAP exploits delaying commitment to

the order in which actions are applicable. This is done to achieve flexibility, reducing the need of backtracking and minimizing the length of the plans by promoting the parallel execution of actions. These changes come at an increase in computational cost, though it allows FLAP to solve more problems than other partial-order planners.

In (Clement, Durfee, and Barrett 2007), multi-agent planning algorithms and heuristics are proposed to exploit summary information during the coordination stage in order to speed up planning. The authors claim that by associating summary information with plans' abstract operators it can ensure plan correctness, even in multi-agent planning, while still gaining efficiency and not leading to incorrect plans. The key idea is to annotate each abstract operator with summary information about all of its potential needs and effects. This process often resulted in an exponential reduction in planning time compared to a flat representation. Their approach depends on some specific conditions and assumptions, and therefore cannot be used in all domains, i.e., it is not a general-purpose system.

Kovacs proposed a recent extension for PDDL3.1 that enables the description of multi-agent planning problems (Kovacs 2012). It copes with many of the already discussed open problems in multi-agent planning, such as the exponential increase of the number of actions, but it also approaches new problems such as the constructive and destructive synergies of concurrent actions. Although only the formalism is provided (it is not yet implemented in any system), the ideas expressed by Kovacs are enticing, making it an interesting candidate to add to DOMAPS planning formalisms.

7 Conclusion

In this dissertation abstract we described the design of a Distributed Online Multi-Agent Planning System (DOMAPS). Specifying each of its main components: *i)* the planning formalism – we introduced the MA-HTN formalism, a multi-agent variation of the traditional single-agent HTN formalism; *ii)* the goal allocation mechanism – by using a contract net protocol, the agents that participate in the planning stage can pre-select the goals that they believe to be more appropriate to them, this pre-planning can cut the planning time considerably in domains with very heterogeneous agents; *iii)* the individual planner – the SHOP2 planner is used in each agent for individual planning, so as to make the most of the HTN-like structure of the plan library in Jason agents; *iv)* the coordination mechanism – employment of social laws to coordinate the agents during run-time in order to avoid possible conflicts made during planning.

Initial experiments and experience with DOMAPS has presented enough positive incentives to pursue solutions for the limitations and to provide improvements for the system overall.

References

Boissier, O.; Bordini, R. H.; Hübner, J. F.; Ricci, A.; and Santi, A. 2011. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*.

Clement, B. J.; Durfee, E. H.; and Barrett, A. C. 2007. Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research (JAIR)* 28:453–515.

Crosby, M.; Jonsson, A.; and Rovatsos, M. 2014. A single-agent approach to multiagent planning. In *21st European Conf. on Artificial Intelligence (ECAI'14)*.

de Weerdt, M., and Clement, B. 2009. Introduction to Planning in Multiagent Systems. *Multiagent Grid Syst.* 5(4):345–355.

desJardins, M. E.; Durfee, E. H.; Ortiz, C. L.; and Wolverton, M. J. 1999. A survey of research in distributed, continual planning. *AI Magazine* 20(4).

Durfee, E. H., and Zilberstein, S. 2013. Multiagent planning, control, and execution. In Weiss, G., ed., *Multiagent Systems 2nd Edition*. MIT Press. chapter 11, 485–545.

Durfee, E. H. 1999. Distributed problem solving and planning. In *Mutliagent systems*. MIT Press. 121–164.

Kovacs, D. L. 2012. A multi-agent extension of pddl3.1. In *Proceedings of the 3rd Workshop on the International Planning Competition (IPC)*, ICAPS-2012, 19–27.

Meneguzzi, F., and De Silva, L. 2013. Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning. *The Knowledge Engineering Review* FirstView:1–44.

Nau, D.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Nissim, R., and Brafman, R. I. 2014. Distributed heuristic forward search for multi-agent planning. *J. Artif. Intell. Res. (JAIR)* 51:293–332.

Sapena, O.; Onaindia, E.; and Torreño, A. 2015. FLAP: applying least-commitment in forward-chaining planning. *AI Commun.* 28(1):5–20.

Shoham, Y., and Tennenholtz, M. 1995. On social laws for artificial agent societies: Off-line design. *Artif. Intell.* 73(1-2):231–252.

Singh, M., and Chopra, A. 2010. Programming multiagent systems without programming agents. In Braubach, L.; Briot, J.-P.; and Thangarajah, J., eds., *Programming Multi-Agent Systems*, volume 5919 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 1–14.

Smith, R. G. 1980. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.* 29(12):1104–1113.

Weerdt, M. D.; Mors, A. T.; and Witteveen, C. 2005. Multi-agent planning: An introduction to planning and coordination. Technical report, Handouts of the European Agent Summer.

Integrating Planning and Recognition to Close the Interaction Loop

Richard G. Freedman
College of Information and Computer Sciences
University of Massachusetts Amherst
freedman@cs.umass.edu

Introduction

In many real-world domains, the presence of machines is becoming more ubiquitous to the point that they are usually more than simple automation tools for one-way interaction. As part of the environment amongst human users, it is necessary for these computers and robots to be able to interact back reasonably by either working independently around them or participating in a task, especially one with which a person needs help. Such interactions are now everywhere ranging from robots around homes and factories to virtual agents in mobile devices, video games, and automated dialogue systems. While interactive robots and computer systems have been implemented for various domains, most are specifically designed for a given domain such as industrial robotics (Levine and Williams 2014; Wurman, D'Andrea, and Mountz 2007), elderly care (Schwenk, Vaquero, and Nejat 2014; Fasola and Matarić 2013), etc. Just as there are domain-independent heuristics that can effectively find optimal solutions for any classical planning problem (Hoffmann and Nebel 2001; Helmert 2006), I introduce a domain-independent approach to performing interaction based on the integration of several research areas in artificial intelligence, particularly planning and plan recognition.

This interactive procedure requires several steps performed indefinitely as a loop: recognizing the user and environment from sensor data, interpreting the user's activity and motives, determining a responsive behavior, beginning to perform the behavior, and then recognizing everything again to confirm the behavior choice and replan if necessary. At the moment, the research areas addressing these steps, activity recognition, plan recognition, intent recognition, and planning, have all been primarily studied independently. However, pipelining each independent process can be risky in real-time situations where there may be enough time to only run a few steps. This leads to a critical question: *how do we perform everything under time constraints*? In this thesis summary, I propose a framework that *integrates these processes* by taking advantage of features shared between them. This includes my current work towards this preliminary system and outlines how I plan to complete the integration for a time-constrained interaction loop.

Background

One of the earliest areas of artificial intelligence, *planning* is the study of automated action selection. Early approaches usually involved representing the world as a list of logic statements and searching for a sequence of actions which would modify the list until it contained the set of goal conditions; the notation used for this is called *STRIPS*. Modern approaches range from improving search over STRIPS to decision theoretic planning with MDPs and its variants to approximation methods to handle uncertainties in the world.

As its inverse problem, *plan recognition* (PR) tries to identify the problem an agent is solving given its observed actions. The actions and problems are usually represented at a higher level such as STRIPS. *Activity recognition* (AR) works at the lower level by interpreting sensor data as higher-level actions. In addition to predicting current activity, *intent recognition* (IR) tries to predict the agent's specific goal or upcoming actions which allows some degree of foresight into the observed agent's behavior. Collectively, these fields of recognition are referred to as *PAIR* and have become a more popular area of research recently, including the topic of a Dagstuhl Seminar (Goldman et al. 2011). Various problems in PAIR are studied in other areas, sometimes under different names, making the literature vast, but they are still studied largely independently or pipelined in most these works.

One notable work which integrated plan and activity recognition was by Levine and Williams (2014) where, for a given plan with branching points based on a human's choice of actions, a robot would select actions to resolve broken causal links resulting from the human's action choice(s). Our approach differs from this work because the given plan provides instructions for the human to follow, but we do not restrict the human with directions. This assumed plan is fine for their intended factory domain, but insufficient for general interaction in any domain. Daily tasks may be intertwined over time or contain noise such as answering a ringing telephone while cooking or cleaning; such domains cannot assume that a human will follow a protocol.

Simultaneous Plan and Activity Recognition

The formulation of a typical recognition problem is as follows: given a sequence $\mathcal{O}$ of observations $o_1, o_2, \ldots, o_n$, determine which task(s) in library L the agent is performing. In AR, each o_i is a sensor reading and L is a set of actions or activities. Supervised machine learning and graphical models are usually used to infer the label in L which best describes $\mathcal{O}$. For PR, each o_i is a STRIPS action and L contains the

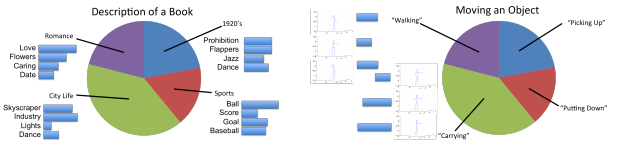


Figure 1: Analogy of Distributions for Topics and Actions

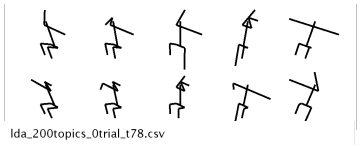


Figure 2: Learned Cluster of Postures Resembling Sitting

set of all tasks. A weighted matching method is often used to compare $\mathcal{O}$ to some of each task's solutions called *plans*, a sequence of STRIPS actions that satisfies the goal conditions. A recent method used to perform this matching is parsing hierarchical task networks, revealing a *parallel between PR and natural language processing* (NLP) (Geib and Steedman 2007). In particular, $\mathcal{O}$ is like a sentence and the breakdown of a complex task into subtasks is like a grammar of valid derivations.

I began the extension of this analogy by considering another problem in NLP: topic modeling. Unlike parsing which determines the underlying structure of a sentence, topic modeling investigates the concepts discussed in a collection of documents by finding clusters of related words called *topics*. For the popular Latent Dirichlet Allocation (LDA) topic model (Blei, Ng, and Jordan 2003), which was previously used for AR by Huỳnh, Fritz, and Schiele (2008), each topic $t \in T$ is simply a distribution over the set of words V and each document $d \in D$ is a distribution over the set of topics T; the respective distributions $\phi_{t \in T} : V \to [0, 1]$ and $\theta_{d \in D} : T \to [0, 1]$ are learned using unsupervised learning to model the training data. For interaction in a variety of domains, an unsupervised approach is more appealing because it is possible to learn a large number of activities in each domain as they are added to the system. Furthermore, LDA's *bag-of-words assumption* where each o_i is independent of the rest of $\mathcal{O}$ was appealing to begin integrating AR and PR due to the mismatch of the two sequence formations. A sensor records over time so that a single action has multiple consecutive o_i for AR, but a single STRIPS action is only one o_i for PR. Figure 1 illustrates how this analogously treats actions like topics. Each word is a sensor reading and the distribution over these readings describes an action while the task of the recording session is represented by the distribution of actions. *Hence inferring a topic with LDA performs AR and the distribution of the collection of inferences enables us to approximate PR simultaneously*. The results from running LDA on a small dataset provided evidence supporting my hypothesis since each topic contained postures resembling simple actions as in Figure 2 (Freedman, Jung, and Zilberstein 2014).

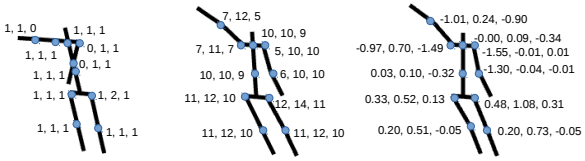


Figure 3: A posture reaching up towards a shelf with granularity 3 (left) and 21 (middle) from the original angles (right).

Representation of RGB-D Sensor Readings For robotics applications, Red, Green, Blue - Depth (RGB-D) sensors have become one of the most common sensors to use due to their present availability, affordability, and growing popularity. They provide a camera image with an overlaid infra-red scan to obtain three-dimensional point clouds of their surroundings, which aids segmentation and associating individual components' directions of motion (Herbst, Ren, and Fox 2013; Zhang, Nakamura, and Kaneko 2015) including a person's entire body as a stick figure rendering (Shotton et al. 2011). Because LDA is designed for inputs whose frequencies resemble words in a natural language, it is important to treat these stick figures in a similar manner. The process of converting data into a form for natural language methods is wordification (Perovšek et al. 2013).

The human posture is composed of fifteen joints via ten rotation matrices; though it is sufficient to create a "vocabulary" of postures as the concatenation of the rotation matrices' Eulerian transformations (roll, pitch, and yaw), the frequency of postures is not desirable. In particular, each wordified posture is in $[-\pi, \pi]^{30}$, which is an uncountably infinite vocabulary with a very small likelihood of duplicates. Zipf's Law states that not only should there be duplicate words in any natural language, but that the frequency of the i^{th} most frequent word is about twice that of the $(i+1)^{\text{th}}$ most frequent word. Hence we make the vocabulary finite and increase the likelihood of duplicates by *discretizing the space* with respect to a *granularity* parameter $g \in \mathbb{N}$, mapping each angle to $[0, g) \cap \mathbb{Z}$ as illustrated in Figure 3. The reduced vocabulary $\{0, 1, \ldots, g-1\}^{30}$ is still large in cardinality for small g, but many poses do not represent feasible body structures. For example, the limitations of each joint's range of motion will not form postures with hyperextended limbs, much the same as many combinations of orthographic letters do not form actual words in a language. Our initial investigation of g showed that increasing it reduces the number of duplicate words and that odd g have more duplicates than even g due to small body motions about the origin.

Extensions for Additional Features I extended LDA to a generative model that also considers the presence of nearby objects and/or global temporal patterns. The set of objects with which the user can interact are represented using a second vocabulary in a separate LDA model, but it shares the same action/topic as the posture at each respective timestep. The temporal relation is captured in the composite topic model (Griffiths et al. 2004) by embedding LDA as a state within a hidden Markov model (HMM), enforcing a syntac-

tical structure with the HMM where *one state is a 'blank' for semantic words/postures/objects derived by LDA*. The improved log-likelihoods of observed task executions with our new topic model variations serve as evidence that the information provided by these two factors are not only independent, but assist disambiguating actions that contain common postures (Freedman, Jung, and Zilberstein 2015). In future work, I will investigate additional variations and their insights for PAIR.

Integration of Planning with Plan Recognition

Ramírez and Geffner (2010) introduced a compilation of PR problems into classical planning problems. It assigned a distribution over sets of goal conditions $\mathcal{G}$ instead of over precomputed plans; they refer to this generalization as a domain instead of a library. Bayes's Rule compares the compiled classical planning problems for each entry of the domain against each other using the most optimal plans with and without $\mathcal{O}$ as a subsequence. This accounts for the probability of the agent solving each task conditioned on its observed actions, considering optimal (shorter) plans to be more likely. While the accuracy for the method is very strong, a temporal plot of the probabilities showed that it only achieved this accuracy towards the *completion of the plan* when the final actions were observed.

While their compilation is excellent when the plan's completing actions are observed, this is not as practical in interactions because the observed agent will likely be almost finished executing a plan by the time the machine can respond. In collaboration with Fukunaga (University of Tokyo), I have proposed two approaches to address this (Freedman and Fukunaga 2015). The first one generates a *dynamic prior* for Bayes's Rule that removes the bias for shorter plans and converges to the true prior over time. The second one counts the number of linearizations of a partially-ordered plan in order to account for the number of optimal plans rather than their length alone.

The updated distribution over $\mathcal{G}$ can be used to aggregate the lists of logic statements which must be true for each goal and *identify the most necessary conditions*. If a set of conditions is shared between the most likely tasks, then satisfying them should be required regardless of the task. Thus a second pass of the planner on a variation of the original classical planning problem should yield a plan that the machine may execute to interact properly even if the recognized task is still ambiguous. In order to consider potential coordination between the observed agent and the machine, we assume that the response problem is a centralized multi-agent planning problem and that the observed agent will perform its actions assigned from that plan. In reality, the agents are decentralized and this synchronization will likely not hold; thus the interaction loop begins again with the recognition steps to determine how the observed agent reacts.

Status of Thesis

I plan to close the interaction loop by completing the integration of these processes. Besides continuing the works above, there are several key remaining tasks. The most important one is bridging the gap between simultaneous PR and AR and the integration of PR with planning. Although it seems trivial because both contain PR, they do not align due to the unsupervised nature of topic models. Recognized actions are clusters of postures (or other forms of sensor data) without annotation while actions in the newer research are assumed to be in STRIPS which is designed by humans. I have begun to identify methods for autonomously extracting features of the postures with the greatest probability mass in each cluster and using them to describe the respective action (Freedman and Zilberstein 2015). In addition to applying this automated feature extraction process to sensor data, I am exploring analogies in topic modeling for natural language data with Wallach (Microsoft Research).

I am also considering the application of constraint optimization to align LDA clusters (the recognized actions) with STRIPS operators using ordering of each $\mathcal{O}$ and these extracted features. Due to the frequency of observations for sensors compared to higher-level actions, $|\mathcal{O}_{\text{sensor}}| \geq |\mathcal{O}_{\text{STRIPS}}|$. So there is not necessarily a bijective mapping between the two sequences; however, a single STRIPS action should be associated with a particular subset of LDA clusters for its respective postures. This means we can evaluate a constraint optimization problem of the following form:

- Assign each variable $s_i \in \mathcal{O}_{\text{sensor}}$ a value $p_j \in \mathcal{O}_{\text{STRIPS}}$
- Preserve sequence ordering with constraints $s_1 = p_1$, $s_{|\mathcal{O}_{\text{sensor}}|} = p_{|\mathcal{O}_{\text{STRIPS}}|}$, and $s_i = p_j \Rightarrow s_{i+1} \in \{p_j, p_{j+1}\}$
- Ensure that each STRIPS action is associated with a limited subset of LDA clusters with minimization constraints $\min \text{Var}(\tau_a)$ for each STRIPS action a where $\tau_a : T \rightarrow [0, 1]$ is a probability distribution over the LDA clusters to which observations of a have been assigned.

Due to its higher-level representation, breaking a single STRIPS action into smaller subactions like a hierarchical task network (HTN) (Erol, Hendler, and Nau 1994) may facilitate the alignment process with additional constraints for each subaction. However, it would also be necessary to perform additional search to find the correct HTN breakdown for the alignment. This introduces new challenges of identifying *heuristics to evaluate snapshots of optimality*. A visualization of our extended generative model for recognition and this HTN alignment is shown in Figure 4.

The integration of other components such as planning and execution have previously been studied in such areas as metareasoning (Russell and Wefald 1991; Zilberstein 2011). After implementing and testing the proposed integrations of recognition and planning, it will be ideal to integrate IR to better predict upcoming actions so that the machine does not interfere with the observed user. For this, I intend to investigate the planning graph (Blum and Furst 1997) and determine how to probabilistically select action nodes which are more likely to be executed. Once these are all in tact, the preliminary interaction loop will be complete and optimization will be necessary to make it usable under realistic time constraints. For example, the work that currently integrates planning and plan recognition runs a classical planner $2|\mathcal{G}|$ times from the same initial state to identify all the plans for

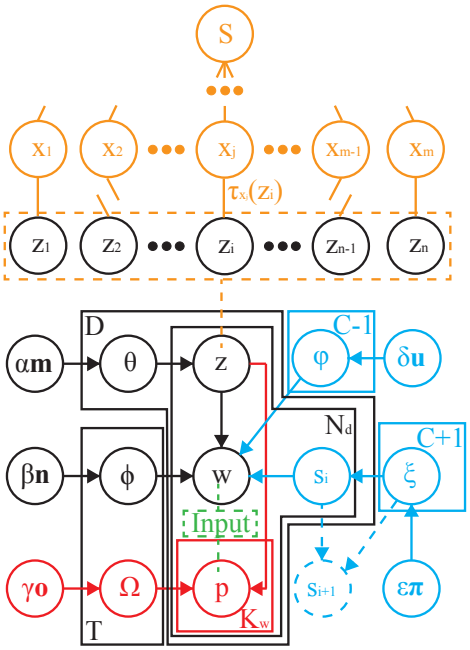


Figure 4: Layout of integrated plan and activity recognition generative model with wordified sensor inputs and objects (green), LDA (black), extension for objects (red), HMM for the temporal extension (blue), and aligned HTN (orange) where S is the sequence of STRIPS actions to break down.

the Bayes's Rule computation, but there is research on finding multiple goals in a single heuristic search so that the classical planner only needs to run one time (Davidov and Markovitch 2006). This avoids redundant expansion of the state space.

References

Blei, D. M.; Ng, A. Y.; and Jordan, M. I. 2003. Latent Dirichlet allocation. *Journal of Machine Learning Research* 3:993–1022.

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1-2):281–300.

Davidov, D., and Markovitch, S. 2006. Multiple-goal heuristic search. *Journal of Artificial Intelligence Research* 26:417–451.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1123–1128.

Fasola, J., and Matarić, M. J. 2013. Socially assistive robot exercise coach: Motivating older adults to engage in physical exercise. In Desai, J. P.; Dudek, G.; Khatib, O.; and Kumar, V., eds., *Experimental Robotics*, volume 88 of *Springer Tracts in Advanced Robotics*. Springer International Publishing. 463–479.

Freedman, R. G., and Fukunaga, A. 2015. Integration of planning with plan recognition using classical planners (extended abstract). In *Proceedings of AAAI 2015 Fall Symposium on AI for Human-Robot Interaction*, 48–50.

Freedman, R. G., and Zilberstein, S. 2015. Automated interpretations of unsupervised learning-derived clusters for activity recognition. In *Workshop on Learning for Human-Robot Collaboration*.

Freedman, R. G.; Jung, H.-T.; and Zilberstein, S. 2014. Plan and activity recognition from a topic modeling perspective. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, 360–364.

Freedman, R. G.; Jung, H.-T.; and Zilberstein, S. 2015. Temporal and object relations in unsupervised plan and activity recognition. In *Proceedings of AAAI 2015 Fall Symposium on AI for Human-Robot Interaction*, 51–59.

Geib, C. W., and Steedman, M. 2007. On natural language processing and plan recognition. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 1612–1617.

Goldman, R. P.; Geib, C. W.; Kautz, H.; and Asfour, T. 2011. Plan Recognition – Dagstuhl Seminar 11141. *Dagstuhl Reports* 1(4):1–22.

Griffiths, T. L.; Steyvers, M.; Blei, D. M.; and Tenenbaum, J. B. 2004. Integrating topics and syntax. In *Proceedings of the Eighteenth Annual Conference on Neural Information Processing Systems*, 537–544.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26(1):191–246.

Herbst, E.; Ren, X.; and Fox, D. 2013. RGB-D flow: Dense 3-D motion estimation using color and depth. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2276–2282.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Huỳnh, T.; Fritz, M.; and Schiele, B. 2008. Discovery of activity patterns using topic models. In *Proceedings of the Tenth International Conference on Ubiquitous Computing*, 10–19.

Levine, S., and Williams, B. 2014. Concurrent plan recognition and execution for human-robot teams. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, 490–498.

Perovšek, M.; Vavpetič, A.; Cestnik, B.; and Lavrač, N. 2013. A wordification approach to relational data mining. In Fürnkranz, J.; Hüllermeier, E.; and Higuchi, T., eds., *Discovery Science*, volume 8140 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 141–154.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 1121–1126.

Russell, S. J., and Wefald, E. 1991. Principles of metareasoning. *Artificial Intelligence* 49(1-3):361–395.

Schwenk, M.; Vaquero, T.; and Nejat, G. 2014. Schedule-based robotic search for multiple residents in a retirement home environment. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2571–2577.

Shotton, J.; Fitzgibbon, A.; Cook, M.; Sharp, T.; Finocchio, M.; Moore, R.; Kipman, A.; and Blake, A. 2011. Real-time human pose recognition in parts from a single depth image. In *Proceedings of the Twenty-Fourth IEEE Conference on Computer Vision and Pattern Recognition*, 1297–1304.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2007. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *Proceedings of the Nineteenth National Conference on Innovative Applications of Artificial Intelligence*, volume 2 of *IAAI'07*, 1752–1759. Vancouver, British Columbia, Canada: AAAI Press.

Zhang, B.; Nakamura, T.; and Kaneko, M. 2015. Adaptive fusion of multi-information based human identification for autonomous

mobile robot. In *Proceedings of the Twenty-fourth IEEE International Symposium on Robot and Human Interactive Communication*.

Zilberstein, S. 2011. Metareasoning and bounded rationality. In Cox, M., and Raja, A., eds., *Metareasoning: Thinking about Thinking*. Cambridge, MA, USA: MIT Press. 27–40.

Dissertation Abstract: Distributed Privacy-preserving Multi-agent Planning

Andrea Bonisoli
Univrsità degli Studi di Brescia
Dipartimento di Ingegneria dell'Informazione
Via Branze, 38 - 25123 Brescia (IT)
andrea.bonisoli@unibs.it

Abstract

Planning is a well known and studied field of Artificial Intelligence. Multi-Agent Planning concerns the construction of plans for a group of autonomous agents that can interact. The aim of multi-agent planning is to automatically find a solution such that, if every agent executes successfully his plan, the environment changes to a goal state. The solution can be found either by centralized or distributed algorithms.

Introduction

The aim of planning, a well known field of Artificial Intelligence, is the automated synthesis of partially ordered sequences of actions, called plans, that can be executed in given settings by one or more agents. A plan is called a solution for a given problem if its execution from an initial known state achieves the problem goals.

Multi-agent planning can be seen as an extension of classical planning and in (De Weerdt and Clement 2009) is defined as "the problem of planning by and for a group of agents". This definition is intentionally general and therefore includes many different approaches. Multi-agent planning can be applied to a wide range of problems, from team of robots involved in space exploration or disaster recovery to logistics chains involving different companies. Whenever there are multiple actors that operate in the setting and they need to decide the best course of action, multi-agent planning can be used to find a solution. It is also worth noting that, although multi-agent planning is not a new research field, many important contributions in this topic are quite recent. One of the main motivations in multi-agent planning is that some or all agents have private knowledge that cannot be communicated to other agents during the planning process and the plan execution.

Problem Definition

Different authors use some slightly different definition of multi-agent planning, however the most common definition of this problem relies on the multi-agent language called `MA-STRIPS`, a minimal extension of the `STRIPS` planning language, which was first described in (Brafman and Domshlak 2008) and then adopted by several authors (Jonsson and Rovatsos 2011; Nissim and Brafman 2012; Brafman and Domshlak 2013; Štolba and Komenda 2013; Štolba, Fišer, and Komenda 2015a). Other definition of the problem are also possible (Torreño, Onaindia, and Sapena 2014; 2015; Bonisoli et al. 2014), nonetheless `MA-STRIPS` is a simple and effective language to represent the *cooperative* multi-agent planning.

Formally in a multi-agent planning task is given a set of k agents $\Phi = \{\varphi_i\}_{i=1}^{k}$ and a 4-tuple $\Pi = \langle P, \{A_i\}_{i=1}^{k}, I, G\rangle$ where:

- P is a finite set of atomic propositions, $I \subseteq P$ encodes the initial state and $G \subseteq P$ encodes the goal conditions, as in the classical planning;
- A_i, for $1 \leq i \leq k$, is the set of actions of the agent φ_i . Every action $a \in A = \bigcup A_i$ is given by its preconditions and effects; every agent has a different set of actions, i.e. $A_i \cap A_j = \emptyset$ if $i \neq j$.

A solution is a partially ordered sequence of actions such that each action in the plan is associated with a single agent. If there is only one agent in the problem, that is n = 1, this definition reduces exactly to a `STRIPS` problem. Therefore, one can see `MA-STRIPS` as a partition of the set of actions of a `STRIPS` problem and assignment of one agent to each partition set. This rather simple extension of the language is easy to understand, but it is quite limited: for example in `MA-STRIPS` it is not possible to define different goals for different agents. The model was also extended for the case of *self-interested* agents (Nissim and Brafman 2013). Another interesting proposal for a standard description language that allow for a more direct comparison between systems and approaches is MA-PDDL (Kovacs 2012), which is an extension of the PDDL language used by the international planning competitions. MA-PDDL is aimed at solving most of the limitations of other multi-agent planning languages. This language can be used to describe many different multi-agent systems and was used during the first Competition of Distributed and Multiagent Planners (Štolba, Komenda, and Kovacs 2015).

Given the partition of actions in `MA-STRIPS`, is is possible to distinguish between *private* and *public* actions and propositions. An atom is *private* for agent φ_i if it is required and affected only by the actions of φ_i. An action of φ_i is *pri-*

vate if all its preconditions and effects are private. All other action are classified as *public*. This definitions are quite important and widely used because are the basis for defining the privacy of the agents.

State of the Art

One of the first distributed algorithms to solve a MA-STRIPS problem is presented in (Brafman and Domshlak 2008). This approach is based on a distributed solver for constraint satisfaction problems: using the notions of public and private actions it is possible to define constraints for the agents and use them to agree on a solution and let each agent to autonomously plan their private part. The experimental results of this algorithm are presented in (Nissim, Brafman, and Domshlak 2010). Although the basic idea was highly innovative, the overall efficiency of the algorithm was insufficient and, therefore, was only able to solve problems of modest size and complexity. In the same work it is defined an upper limit to the complexity resolutive for the problems MA-STRIPS that depends exponentially on two parameters that quantify the level of coupling of the system.

To increase scalability than the number of agents involved in (Jonsson and Rovatsos 2011) is proposed an approach with iterative refinement of the plans: each agent involved may in turn change their plan to update it according to the actions planned by other agents. In this way a process is obtained which slowly converges towards a solution plan for the problem. Unfortunately, this convergence is not guaranteed in all cases, as it can not formally prove the optimality of the solutions produced by the algorithm. However it uses in it known technologies and planners available oof the shelf and can therefore take advantage of the latest innovations in the field of research into classical planning. It can also be used to solve problems with non-cooperative agents, but only if it is not required to keep private the plans of the agents.

In contrast in (Nissim and Brafman 2012; 2013) is presented the extension to the case of multi-agent of the known search algorithm A$\star$, called MAD-A$\star$. This new algorithm maintains the properties known from classic case if the heuristic function used is permissible and also ensures a level of privacy to agents involved which the authors define as *weak privacy*. Moreover, in (Nissim and Brafman 2014) is also described an additional feature of MAD-A$\star$ which allows him to prune the search tree of a larger number of nodes than classic algorithm. Finally in (Brafman 2015) describes a modification to the algorithm that allows it to further increase the level of privacy and exchanging fewer messages between agents, but unfortunately for this version are not present experimental results.

Instead in (Torreño, Onaindia, and Sapena 2012; 2013; 2014; 2015) is presented an algorithm based on forward planning with partial ordering and exchange of incomplete plans. In this case the problem of the proposed model is different than that described in MA-STRIPS, however, the two models are comparable and the most recent implementation of this algorithm is able to solve the problems described with MA-PDDL. It is noteworthy that the privacy of agents is kept obscuring a part of the partial plans that are exchanged during the planning of the agents.

Conversely in (Maliah, Shani, and Stern 2014) is presented the algorithm GPPP which uses explicitly two different phases to identify a solution plan. During the first phase the agents want to identify a joint coordination scheme through a high-level planning of a relaxed problem using only the public actions of the problem. In the second phase, each agent individually shall seek a local plan that can support public actions agreed in coordination shcema. It is evident that, since the high-level planning has been carried out on a relaxed problem, it is possible that during the second phase some of the agents are not able to find a viable solution. In this case the algorithm executes again the first phase to determine a different coordination scheme.

Finally in (Tozicka, Jakubuv, and Komenda 2014; 2015) describes a new and recent approach based on finite state machines. This algorithm called PSM uses finite automata to represent a set of plans and then use public screenings and intersection of automata to compute a solution. Since public projections do not contain private information, these are exchanged between agents that have generated a plan and if the intersection between the projections received from other agents and its plan is not empty, then the plan is a solution. The planner based on this algorithm has shown excellent results in the competition for distributed planners, in particular in the fully distributed track.

It is also important to note the studies regarding the heuristic evaluation functions because of their importance during the research phase: a more accurate heuristic could bring significant performance gains for algorithms that use it. For this reason, in (Štolba, Fišer, and Komenda 2015b) shows a comparison between the performance of different heuristics multi-agent. Many of those presented are in fact an adaptation of heuristics known for classical planning adapted to the case multi-agent. Most of these heuristics is an adaptation of the very heuristic notes taken by the famous classic planner Fast-forward described in (Hoffmann and Nebel 2001). It is for example the case of (Štolba and Komenda 2013; 2014) which distributes between agents graphs of relaxed schedule used by fast-forward. The authors Torreño, Onaindia, and Sapena on the contrary using an approach based on the latest graphs domain transition, although in their last work (Torreño, Onaindia, and Sapena 2015) show that a hybrid approach between the two different heuristics may result on average in more accurate heuristic. In (Maliah, Shani, and Stern 2014) is instead presented a multi-agent heuristi based on landmark that can also to maintain the privacy of the agents, while in (Štolba, Fišer, and Komenda 2015a) describes another heuristic based on lankmark but admissible. In (Maliah, Shani, and Stern 2015) a new heuristic based on pattern database is proposed and shoed to be more effectiv on some domains. It should however be noted that the distributed heuristic algorithms can be substantially different from the classic ones that inspired them and in general could also get different results with respect to their central counterparts.

Privacy-preserving Multi-agent Planning

One fundamental problem of MA-STRIPS is that it cannot express the privacy of the agents beyond the definitions of public and private facts and actions and does not fully guarantee the privacy of the involved agents when at least one public proposition is confidential (i.e., it should be kept hidden from some agent). For example a proposition shared by two agents should be public for every other involved agent.

Therefore in this section, we propose a more general model that preserves the privacy of the involved agents. The model of multi-agent planning that is most similar to the one we propose here is the model adopted by MAP-POP (Torreño, Onaindia, and Sapena 2012). This model was first presented in (Bonisoli et al. 2014).

A *privacy-preserving multi-agent planning problem* for a set of agents $\Sigma = \{\alpha_i\}_{i=1}^n$ is a tuple $\langle \{A_i\}_{i=1}^n, \{F_i\}_{i=1}^n, \{I_i\}_{i=1}^n, \{G_i\}_{i=i}^n, \{M_i\}_{i=1}^n \rangle$ where:

- A_i is the set of actions agent α_i is capable of executing, and such that for every pair of agents α_i and α_j, $A_i \cap A_j = \emptyset$;
- F_i is the set of relevant facts for agent α_i;
- $I_i \subseteq F_i$ is the portion of the initial state relevant for α_i;
- $G_i \subseteq F_i$ is the set of goals for agent α_i;
- $M_i \subseteq F_i \times \Sigma$ is the set of messages agent α_i can send to the other agents.

Facts and actions are literals and pair $\langle \mathit{Pre}, \mathit{Eff} \rangle$, respectively, where *Pre* is a set of positive literals and *Eff* is a set of positive or negative literals. Let $X+/X-$ denote the positive/negative literals in set X, respectively. Let $\mathcal{G}$ be the graph induced by $\{M_i\}_{i=1}^n$, where nodes represent agents, and edges represent possible information exchanges between agents; i.e., an edge from node α_i to node α_j labelled p represents the agent α_i's capability of sending p to agent α_j. In order to have well-defined sets $\{M_i\}_{i=1}^n$, $\forall \alpha_i, \alpha_j \in \Sigma$, $\forall p$ s.t. $p \in F_i$ and $p \in F_j$, there should be a path in $\mathcal{G}$ from the node representing α_i to the node representing α_j formed by edges labelled p, if $p \in I_i$, or $\exists a \in A_i \cdot p \in \mathit{Eff}+(a)$, or $\exists a \in A_i \cdot p \in \mathit{Eff}-(a)$.

A plan for a multi-agent planning problem is a set $\{\pi_i\}_{i=1}^n$ of n single-agent plans. Each single agent plan is a sequence of happenings. Each happening of agent α_i consists of a (possibly empty) set of actions of α_i, and a (possibly empty) set of exogenous events. Exogenous events are facts that become true/false because of the execution of actions of other agents; in this sense, these events cannot be controlled by agent α_i. Formally, $\pi_i = \langle h_i^1, \ldots, h_i^l \rangle$, $h_i^j = \langle A_i^j, E_i^j \rangle$, $A_i^j \subseteq A_i$, $E_i^j \subseteq \bigcup_k F_k$, for $i = 1 \ldots n$, $j = 1 \ldots l$, $k \in \{1, \ldots, i-1, i+1 \ldots, n\}$.

The execution of plan π_i generates a state trajectory, $\langle s_i^0, s_i^1, \ldots, s_i^l \rangle$, where $s_i^0 = I_i$, and a sequence of messages, $\langle m_i^1, \ldots, m_i^l \rangle$, each of which is a set of literals. At planning step j agent α_i sends literal $p/\neg p$ if either α_i executes an action that makes p true/false or α_i receives the message that lets the agent know p becoming true/false. In this latter case, α_i forwards the received message $p/\neg p$ to the agents it is connected to. For every planning step, the forwarding is repeated $n-1$ times so that, if sets $\{M_i\}_{i=1}^n$ are well-defined, every agent α_k such that $p \in F_k$ is advised that p becomes true or false (the length of the shortest path between any pair of nodes in the graph induced by $\{M_i\}_{i=1}^n$ is at most $n-1$). At planning step j agent α_i sends literal $p/\neg p$ if either α_i executes an action that makes p true/false or α_i receives the message that lets the agent know p becoming true/false. In this latter case, α_i forwards the received message $p/\neg p$ to the agents it is connected to. For every planning step, the forwarding is repeated $n-1$ times so that, if sets $\{M_i\}_{i=1}^n$ are well-defined, every agent α_k such that $p \in F_k$ is advised that p becomes true or false (the length of the shortest path between any pair of nodes in the graph induced by $\{M_i\}_{i=1}^n$ is at most $n-1$).

Formally, state s_i^j and message m_i^j are defined as follows, for $j = 1 \ldots l$ and $k = 1 \ldots i-1, i+1 \ldots n$.

$$s_i^j = s_i^{j-1} \cup \bigcup_{a \in A_i^j} \mathit{Eff}+(a) \cup E+_i^j \setminus \bigcup_{a \in A_i^j} \mathit{Eff}-(a) \setminus E-_i^j;$$

$$m_i^j = \bigcup_k \mathit{sm}+_{i \to k}^j(n-1) \cup \bigcup_k \mathit{sm}-_{i \to k}^j(n-1), \text{ with}$$

$$\mathit{sm}+_{i \to k}^j(t) = \Big\{ \langle p, \alpha_k \rangle \mid \langle p, \alpha_k \rangle \in M_i, \; p \in \bigcup_{a \in A_i^j} \mathit{Eff}+(a) \cup \mathit{rm}+_i^j(t-1) \Big\},$$

$$\mathit{sm}-_{i \to k}^j(t) = \Big\{ \langle \neg p, \alpha_k \rangle \mid \langle p, \alpha_k \rangle \in M_i, \; p \in \bigcup_{a \in A_i^j} \mathit{Eff}-(a) \cup \mathit{rm}-_i^j(t-1) \Big\},$$

$$\mathit{rm}+_i^j(t) = \Big\{ p \mid \langle p, \alpha_i \rangle \in \bigcup_k \mathit{sm}+_{k \to i}^j(t) \Big\},$$

$$\mathit{rm}-_i^j(t) = \Big\{ p \mid \langle \neg p, \alpha_i \rangle \in \bigcup_k \mathit{sm}-_{k \to i}^j(t) \Big\},$$

$$\mathit{rm}+_i^j(0) = \mathit{rm}-_i^j(0) = \emptyset.$$

Intuitively, for planning step j, $\mathit{sm}+_{i \to k}^j(t)/\mathit{sm}-_{i \to k}^j(t)$ is the set of positive/negative literals that at the t-th forwarding step ($t = 1 \ldots n-1$) agent α_i sends to agent α_k; $\mathit{rm}+_i^j(t)/\mathit{rm}-_i^j(t)$ is the set of positive/negative literals that at the t-th forwarding step agent α_i receives. Note that propositional planning assumes that at every planning step the execution of actions is instantaneous, and hence the information exchanges also happens instantaneously.

We say that the single-agent plan π_i is *consistent* if the following conditions hold for $j = 1 \ldots l$ and $t = 1 \ldots n-1$:

(1) $E+_i^j = \bigcup_t \mathit{rm}+_i^j(t)$, $E-_i^j = \bigcup_t \mathit{rm}-_i^j(t)$;

(2) $\forall a, b \in A_i^j \cdot \mathit{Pre}(a) \cap \mathit{Eff}-(b) = \mathit{Pre}(b) \cap \mathit{Eff}-(a) = \emptyset$;

(3) $\forall a, b \in A_i^j \cdot \mathit{Eff}+(a) \cap \mathit{Eff}-(b) = \mathit{Eff}+(b) \cap \mathit{Eff}-(a) = \emptyset$;

(4) $\forall a \in A_i^j, \forall e \in E^-_i \cdot \mathit{Pre}(a) \cap e = \emptyset = \mathit{Eff}^+(a) \cap e = \emptyset$.

Basically, (1) asserts that at planning step j all the exogenous events for agent α_i are the positive/negative literals α_i receives during the information exchange, i.e., (1) guarantees that these events are generated by some other agent; (2) and (3) assert that at planning step j agent α_i executes no pair of mutually exclusive actions; finally, (4) asserts that at planning step j agent α_i executes no action that is mutex with some action executed by other agents.

Let $\langle s_i^0, s_i^1, \ldots, s_i^l \rangle$ be the state trajectory generated by single-agent plan π_i. Plan π_i is executable if $\mathit{Pre}(a) \subseteq s_i^{j-1}$, $\forall a \in A_i^j, j = 1 \ldots l$. Plan π_i is valid for agent α_i if it is executable, consistent, and achieves the goals of agent α_i, i.e., $G_i \subseteq s_i^l$. A multi-agent plan $\{\pi_i\}_{i=1}^n$ is a solution of the multi-agent privacy-preserving planning task if single-agent plan π_i is valid for agent α_i, for $i = 1 \ldots n$.

The main difference with existing models to multi-agent planning, like (Torreño, Onaindia, and Sapena 2012), is related to sets $\{M_i\}_{i=1}^n$ and the purpose for which agents use them. Essentially, M_i determines the messages agent α_i can generate during the execution of its plan, that can be sent to other agents without loss of privacy.

Conclusion

Multi-agent planning is an open field of research as many new contributions in recent years have showed and there are still some open issues and challenges to address. First of all, many theoretical properties of some settings of multi-agent planning are not well known. For example it is still unknown the actual complexity of different settings or what make them so difficult. Also, while the theoretical properties of multi-agent systems are well studied in the multi-agent system community, the relation to planning is not a well studied topic and further research work may improve the understanding of the multi-agent planning problem.

Furthermore, while privacy issues are strong reasons for using distributed algorithms, the definition of privacy in multi-agent planning is debated, e.g., what agents should kept private information (state variables, actions, goals) and what minimal information they should exchange in order to be able to construct a joint plan remain an open question. While the distinction between public and private fluents and actions is a first step towards the definition of privacy, it is too weak for many settings not involving cooperative agents. It is unknown whether partial observability can cope with the privacy issues.

References

Bonisoli, A.; Gerevini, A. E.; Saetti, A.; and Serina, I. 2014. A privacy-preserving model for the multi-agent propositional planning problem. *Distributed and Multi-Agent Planning* 25–29.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*, 28–35.

Brafman, R. I., and Domshlak, C. 2013. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence* 198:52–71.

Brafman, R. I. 2015. A privacy preserving algorithm for multi-agent planning and search. *Joint Conference on Artificial Intelligence (IJCAI 2015)* 1530–1536.

De Weerdt, M., and Clement, B. 2009. Introduction to planning in multiagent systems. *Multiagent and Grid Systems* 5(4).

Hoffmann, J., and Nebel, B. 2001. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 253–302.

Jonsson, A., and Rovatsos, M. 2011. Scaling up multiagent planning: A best-response approach. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling*.

Kovacs, D. L. 2012. A multi-agent extension of pddl3. *WS-IPC 2012* 19.

Maliah, S.; Shani, G.; and Stern, R. 2014. Privacy preserving landmark detection. In *Proceedings of ECAI*, volume 14.

Maliah, S.; Shani, G.; and Stern, R. 2015. Privacy preserving pattern databases. *Distributed and Multi-Agent Planning (DMAP-15)* 9.

Nissim, R., and Brafman, R. I. 2012. Multi-agent a* for parallel and distributed systems. In *Proceedings of the Workshop on Heuristics and Search for Domain-Independent Planning*, 42–51. ICAPS.

Nissim, R., and Brafman, R. I. 2013. Cost-optimal planning by self-interested agents. In *AAAI*.

Nissim, R., and Brafman, R. 2014. Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research* 51(1):293–332.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, 1323–1330. International Foundation for Autonomous Agents and Multiagent Systems.

Štolba, M., and Komenda, A. 2013. Fast-forward heuristic for multiagent planning. In *Proceedings of Distributed and Multi-agent Planning (DMAP) Workshop of 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, volume 120.

Štolba, M., and Komenda, A. 2014. Relaxation heuristics for multiagent planning. In *24th International Conference on Automated Planning and Scheduling (ICAPS)*, 298–306.

Štolba, M.; Fišer, D.; and Komenda, A. 2015a. Admissible landmark heuristic for multi-agent planning. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*.

Štolba, M.; Fišer, D.; and Komenda, A. 2015b. Comparison of rpg-based ff and dtg-based ff disrtibuted heuristics. *Proceedings of Distributed and Multi-Agent Planning Workshop(DMAP-15)* 77.

Štolba, M.; Komenda, A.; and Kovacs, D. L. 2015. Competition of distributed and multiagent planners, `http://agents.fel.cvut.cz/codmap/`.

Torreño, A.; Onaindia, E.; and Sapena, Ó. 2012. An approach to multi-agent planning with incomplete information. *Proceedings of ECAI*.

Torreño, A.; Onaindia, E.; and Sapena, Ó. 2013. Fmap: a heuristic approach to cooperative multi-agent planning. In *Proceedings of DMAP Workshop of ICAPS*, volume 13, 84–92.

Torreño, A.; Onaindia, E.; and Sapena, Ó. 2014. Fmap: Distributed cooperative multi-agent planning. *Applied Intelligence* 41(2):606–626.

Torreño, A.; Onaindia, E.; and Sapena, Ó. 2015. Global heuristics for distributed cooperative multi-agent planning. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*.

Tozicka, J.; Jakubuv, J.; and Komenda, A. 2014. Generating multi-agent plans by distributed intersection of finite state machines. In *Proceedings of ECAI*.

Tozicka, J.; Jakubuv, J.; and Komenda, A. 2015. On internally dependent public actions in multiagent planning. *Distributed and Multi-Agent Planning (DMAP-15)* 18.

Planning with Concurrent Execution

Bence Cserna
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
bence@cs.unh.edu

Abstract

In real world planning applications such as planning for robots or video games, time for decision making is often limited. Unlike classical search, real-time search algorithms are applicable in these domains due to their bounded response time. This dissertation addresses the setting in which planning and execution occur concurrently using real-time heuristic search. I advance this area in three ways.

Real-time planning algorithms have to return the next action for the agent within a strict time bound. I show how to adapt previously-proposed methods, which depend on unrealistic assumptions, to allow the use of wall clock time bounds. Second, in real-time search partial plans are generated until the goal state is found. It is not clear whether the agent should commit to one action along the path to best explored node or all the way to the search frontier. We investigate the use of metareasoning algorithms to decide how many actions to commit to. Third, I will extend the application of real-time search algorithms to stochastic domains.

Introduction

In real-time planning domains where human interaction is involved, it is often desirable to reach the goal state as fast as possible. Consider a path-finding domain in which an agent has to navigate to a goal location while the user waits for the execution to be completed.

The performance of classical heuristic search algorithms is often measured by the solution cost, the overall search time and the expansion count. These classical heuristic search metrics are not sufficient for such real-time domains. In problems where the agent has limited time to act the performance of the planning algorithm is measured in goal achievement time (GAT). The GAT measures the wall time from the start of the search to the completion of the task.

Our primary focus is to advance real-time search in three different ways.

Real-time search algorithms operate in planning episodes. After each episode, the planner returns an action or a set of actions until the goal state is expanded. To satisfy the real-time property, the time to complete an episode is bounded.

The current state-of-the-art real-time search, Dynamic $\hat{f}$ provides responsiveness by limiting the number of explored states in the planning episodes(Kiesel, Burns, and Ruml 2015). The time bound of the algorithm is expressed in node expansions. According to the best of the author's knowledge none of the real-time heuristic search algorithms utilize wall time-bound nor are they feasible for such setup. However, in practical applications, the time bound is expressed in wall time, not in node expansion. We propose to use wall time to bound the response time of real-time planners.

In each episode, the planner explores the state space knowing the agent's current location and decides how far it should go in the explored space. Given the framework where the agent acts in parallel to the planning, the time the agent spends on executing actions establishes the time bound for the next planning iteration. The time limit given to the planner determines how far it can look ahead in the search space. More time allows the planner to explore more states and make a more informed decision. It is not clear how many actions should the agent commit to along the path to the best explored state on the search frontier. The early real-time search algorithms only committed to one action, but the leading algorithm, Dynamic $\hat{f}$ sends the agent all the way to the search frontier. We investigate the use of metareasoning algorithms to dynamically decide how many actions to commit to.

Additionally, unlike in classical benchmarks problems, acting in the real world is often stochastic. Consider an environment where a robot has to navigate to a goal location. The location of the agent after taking an action is nondeterministic. The precise location is unknown the planner can only estimate. The leading real-time search algorithms do not consider uncertainty, thus are not applicable to such domains. We propose to extend the existing real-time planning algorithms to domains with uncertainty.

Time bounded real-time search

The first real-time heuristic search algorithms were proposed by Korf in 1990. Since then the area of real-time search has been evolved and several improvements have been proposed. Before discussing the improvements, first we review the most relevant algorithms to our work.

Previous work

A significant drawback of using A* on a situated agent is that the algorithm has to find a complete solution path before making a commitment to the first action. The optimal first move cannot be guaranteed until a complete solution is found.

Learning Real-time A* (LRTA*) is a suboptimal real-time search algorithm (Korf 1990). Unlike A*, LRTA* only considers a local search problem. To select an action LRTA* chooses the successor state that has the lowest sum of cost and cost-to-goal estimate. It commits to one action and repeats the search from the selected successor state. The heuristic value of the neighboring states is augmented by a depth bounded lookahead. The action selection time has a theoretical upper bound due to the depth bounded lookahead.

Local Search Space - LRTA* (LSS-LRTA*) is an improved version of LRTA* (Koenig and Sun 2008).

A search episode of LSS-LRTA* consists of two phases: exploration and learning. In the exploration phase, the algorithm expands the search tree from the agent's current location. This exploration is limited by a predefined expansion bound. The states explored in each episode represent the local search space. While LRTA* selects the best neighboring state and commits to one action per search episode, LSS-LRTA* moves the agent all the way to the search frontier.

After the expansion, LSS-LRTA* uses a learning phase to propagate back the heuristic values from the search frontier in order to help the agent escape the local minima of heuristic values.

Dynamic $\hat{f}$ is a variant of LSS-LRTA* with two improvements. Instead of using and admissible heuristic to explore the local search space, Dynamic $\hat{f}$ utilizes an unbiased inadmissible heuristic function (Kiesel, Burns, and Ruml 2015). In addition, Kiesel, Burns, and Ruml proposed a technique to translate time duration to expansion counts to make the algorithm feasible for practical domains with real-time requirements. Dynamic $\hat{f}$ uses a lookup table based on the available time for the upcoming planning episode. The table is populated by offline learning and contains a predefined set of expansion limits along with the corresponding measured mean planning time.

Real-time search using wall clock bounds

In domains where the time for decision making is limited, the responsiveness of the planning algorithm is essential. We assume the planner algorithm is running on an operating system that does not provide real-time guarantees, thus the running time of an expansion cannot be approximated precisely.

On such systems the state expansion times could be inconsistent, therefore the duration of an episode of an expansion count limited real-time search algorithm is unpredictable, while the bounded response time is the primary requirement.

We propose a real-time variant of LSS-LRTA* that uses real-time measurements to provide more consistent planning episode duration. This could be achieved by replacing the expansion limit with a time limit. The proposed real-time algorithm assesses the time bound after every expansion.

An episode of LSS-LRTA* consists of two phases: exploration and learning. LSS-LRTA* does not account for the time of the learning phase, thus it is not limited by the expansion bound. Dynamic $\hat{f}$ is more sophisticated. It includes both the planning and the execution phases in the time bound by measuring the running time of the episodes. However, these estimates are calculated offline, therefore, they are not adaptive to the changes in the execution environment.

During the exploration phase of LSS-LRTA* or Dynamic $\hat{f}$, the planner expands the explored state space with new states until the time or expansion bound is reached. In the learning phase, the planner propagates back the heuristic values of the search frontier to local search space that were explored in the preceding exploration phase until every state is updated or the bound is reached.

The exploration phase is naturally ended by reaching the bound. However, the learning step is considered incomplete if only a subset of states were updated in the local search space. During the learning phase, the planner increases the heuristic values of the states in the local search space by propagating back the most relevant values from the search frontier. When the time bound is reached during the learning phase the propagation is terminated, leaving the planner in an inconsistent state. The partially updated local search space could be misleading due to the inconsistencies in the heuristic values. The planner would be biased towards the areas that were not updated, therefore, the results of an incomplete learning step should not be used.

The original LSS-LRTA* algorithm does not include the cost of learning in the episode bound due to the fact that it is faster to complete the learning phase than a corresponding exploration phase.

We propose to reverse the order of the learning and exploration in the episodes. The reverse order provides a better chance for the learning phase to complete. After updating the values of the local search space, the exploration phase can utilize the remaining time for the episode. The exploration time and the following learning phase duration are related. More exploration corresponds to longer learning time. When the learning step consumes most of the time in the episode the exploration step has less time to expand new states, thus, the learning step in the following episode will complete faster and leave more time for the exploration.

The proposed method makes LSS-LRTA* and Dynamic $\hat{f}$ feasible for practical real-time domains where the time bound is given in wall time.

A flexible commitment strategy

One of the central issues of real-time heuristic search with a situated agent is the relation between the agent and the plan. How far should the planner look ahead before committing to an action? Should it commit to only one action like LRTA* or should it be less conservative and commit to the

best known state on the search frontier (LSS-LRTA*, Dynamic $\hat{f}$).

O'Ceallaigh and Ruml presented the first real-time search algorithm, Mo'RTS, that incorporates metareasoning to decide when to act, and how many actions to commit to (O'Ceallaigh and Ruml 2015). They showed the Mo'RTS can improve the GAT of leading real-time search algorithms, LSS-LRTA* and Dynamic $\hat{f}$. However, there are flaws that were acknowledged by the authors. First, the Mo'RTS is bounded by node expansions instead of using wall time bounds as LSS-LRTA*. The decision-making time was not included in the metric, thus, the effect of the algorithm on the GAT is not clear.

More importantly, the work of O'Ceallaigh and Ruml, while taking a principled approach to the question of whether or not the agent should commit to an action or deliberate further, takes an ad hoc approach to the question of how many actions the agent should commit to. I will show how, in fact, the second issue can be answer by appealing to the first. By showing that the metareasoning overhead of Mo'RTS can be made small in practice, I will allow the agent to consider the question of whether or not to commit very frequently — at multiple points during a single planning iteration, in fact. This will allow the agent to naturally decide how much of the plan prefix to which to commit, solving the commitment length problem without recourse to an ad hoc method.

The application of meta-reasoning to problems such as plan commitment has shown great promise but has not been fully explored. This dissertation aims to utilize such metareasoning techniques while relaxing the strong assumptions of Mo'RTS.

Extending to stochastic domains

A* based search algorithms are tailored to handle domains where the actions are deterministic. Thus, real-time search algorithms such as LSS-LRTA* and Dynamic $\hat{f}$ are not designed to handle stochastic domains, as plans generated by these algorithms assume certain state transitions.

Markov Decision Processes (MDPs) provide a popular framework for planning problems with uncertainty. Real-time dynamic programming (RTDP) is a generalization of Korf's LRTA* to stochastic domains (Barto, Bradtke, and Singh 1995). Unlike Dynamic $\hat{f}$, that explores only the local search space, RTDP explores states with simulated sequences from the agent's initial location. However, just as with traditional real-time, there has been little consideration of explicitly optimizing GAT. We intend to extend previous work, such as Sanner's Bayesian RTDP (Sanner et al. 2009), using ideas from Mo'RTS to show that MDPs can provide an agile real-time framework for situated agents planning under time pressure.

Conclusion

The primary focus of this dissertation is to overcome the existing issues of real-time heuristic search and to make real-time search actually real-time, while minimizing the goal achievement time. Furthermore, it aims to extend the scope of real-time heuristic search to handle a wider range of real world applications. My dissertation will advance the science of planning with concurrent execution in three ways. First, the current real-time search algorithms are not suited for planning in a real world environment in which the time for decision making is limited in wall time, since they provide responsiveness by limiting the number of expansions. Second, in a setting where the planning is concurrent with the execution, it is not clear whether the agent should commit to one action along the path to best explored node or all the way to the search frontier. Mo'RTS proposed an adaptive metareasoning technique to decide whether the agent should deliberate further, and I will adapt this method to naturally handle the length of commitment. Third, I will extend these ideas to planning under uncertainty.

References

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artif. Intell.* 72(1-2):81–138.

Kiesel, S.; Burns, E.; and Ruml, W. 2015. Achieving goals quickly using real-time search: Experimental results in video games. *Journal of Artificial Intelligence Research* 123–158.

Koenig, S., and Sun, X. 2008. Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems* 18(3):313–341.

Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence* 42(2-3):189–211.

O'Ceallaigh, D., and Ruml, W. 2015. Metareasoning in real-time heuristic search. In *Eighth Annual Symposium on Combinatorial Search*.

Sanner, S.; Goetschalckx, R.; Driessens, K.; and Shani, G. 2009. Bayesian real-time dynamic programming. In Boutilier, C., ed., *{IJCAI} 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 1784–1789.

Session 3

Temporal Planning

Mixed Discrete-Continuous Planning with Complex Behaviors

Enrique Fernandez-Gonzalez
Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Laboratory
32 Vassar Street, Building 32-224, Cambridge, MA 02139
efernan@mit.edu

Abstract

Over the last few years, we have witnessed a tremendous increase in the capabilities of robots. However, robots are still largely teleoperated by humans or controlled by scripts written by experts in advance, in a time-consuming and costly manner. Many practical applications require more autonomous robots, but the current state of the art in planning is not well suited for this task.

This thesis aims to fulfill this need by developing a mixed discrete-continuous temporal planner, *Scotty*, capable of reasoning with complex robot behaviors and accepting high level temporally extended mission goals. Scotty leverages the proven performance of heuristic forward search for symbolic planning with the versatility of trajectory optimization for resolving complex non-linear robot behaviors.

Introduction

Our generation is witnessing a revolution in robotics. Over the last decade we have seen tremendous improvements in robot hardware, perception and control. Robots have transitioned from being, in general, expensive repetitive machines in automotive factories or relatively simple experiments with limited capabilities in research labs, to sophisticated machines able to walk, run, jump, fly or even drive autonomously on highways.

All these recent achievements are due to advances in hardware, perception, control, and the availability of small factor high performance computing power. However, as impressive as these robots are, most of them are either completely scripted in advance, or teleoperated by humans. This is works well for some applications. However, these approaches for operating robots will not be sufficient for many important robotic applications, such as robotic exploration of the Solar System. We need robots able to operate in remote hostile environments in which the availability of direct fast communication with human operators cannot be assumed. Such missions will require a more advanced level of autonomy than what we currently have.

To this day, we are largely not doing any automated planning with robots, as the current state of the art does not fulfill this need. The activity planning community has made impressive advances in symbolic planning, especially since the introduction of heuristic forward search. However, most activity planners, only reason with discrete conditions and effects. Some have been extended to consider limited forms of continuous linear time-evolving effects (Coles et al. 2012; 2010), but still focus mainly on logistic or planning competition problems and are unable to handle the more complicated non-linear effects required to model robot dynamics.

On the other hand, the AI community has expressed an increasing interest in the joint problem of activity and motion planning. Many interesting approaches have emerged over the last few years (Srivastava et al. 2014; Cambon, Alami, and Gravot 2009; Garrett, Lozano-Pérez, and Kaelbling 2014). However, most of these approaches focus on the manipulation problem, which is hard and interesting but requires specific assumptions, and often neglect dynamics, as they are not needed to model common manipulation scenarios.

Finally, we have lately seen great advances in the control of complex underactuated robots, in part thanks to the success of trajectory optimization. There are many examples of successful robot behaviors being implemented using this approach that were impractical to solve in a reasonable amount of time just a few years ago (Fallon et al. 2015; Dai, Valenzuela, and Tedrake 2014). To our knowledge there are no planners able to reason with these complex behaviors.

This thesis aims to address this need by developing a mixed discrete-continuous temporal planner, **Scotty**, able to plan with complex robot behaviors. The Scotty planner leverages the proven performance of heuristic forward search for symbolic planning and the recent advances in trajectory optimization to reason with complex robot behaviors requiring non-linear dynamics.

Problem Description

Scotty is a hybrid temporal planner and, as such, its problem description inherits the main elements from the activity planning literature. Scotty's problem statement is similar to PDDL2.1(Fox and Long 2011) in that the state of the system is given by true/false propositions and values to numeric state variables, and that activities have preconditions (*at start*, *over all*, *at end*) and effects (*discrete at start* and *at end* and also *continuous*). Scotty's problem statement differs from PDDL2.1 in that it allows more expressive goal specifications and more complex activities with non-linear

dynamics and conditions that we call **behaviors**.

Scotty takes as inputs an *initial state*, a *planning domain* and *time-evolved goal representation* and produces a *plan*. The input to Scotty is a tuple $\langle I, A, G \rangle$ where

- I is the initial state of the system at $t = 0$. The state of the system at time t, $X(t)$ is given by:
  - The set of propositions that hold at time t
  - An assignment to all the continuous state variables of the system, $x_i(t)$

  the initial state of the system is then $X(0)$
- D is the planning domain that defines the allowed *activities* and *behaviors*
- G is the temporally-extended goal specification

The output of Scotty is a temporal plan satisfying the goal specification.

Temporally-extended goals

The goal specification in the classic temporal planning problem simply consists of the set of proposition that need to hold at the end of the plan. This definition is static: it does not matter what happens between the start and the end of the plan as long as the objectives are satisfied at the end. However, in most real world problems objectives evolve over time and there may be temporal restrictions between them.

We will use Qualitative State Plans (QSPs)(Li and Williams 2011; Léauté and Williams 2005; Hofmann and Williams 2006) to represent these temporally-extended goals. QSPs use episodes and temporal constraints to specify the users' objectives and requirements on states, resources and time. Formally, a QSP is represented as a 3-tuple $< E, EP, TC >$, where

- E is a set of events. Each event $e \in E$ can be assigned a non-negative real value which denotes a point in time.
- EP is a set of episodes. Each episode specifies an allowed state trajectory between a starting and an ending event. They are used to represent the state constraints. An episode, ep, is a tuple $< e_S, e_E, l, u, sc >$ where:
  - e_S and e_E in ep are the start and end events of the state trajectory.
  - l and u are the lower and upper bounds on the temporal duration of the episode.
  - sc is a state constraint that must hold true over the duration of the episode. The state constraint sc can be either a discrete predicate, or a condition over the state variables.
- TC is set of simple temporal constraints. It represents the temporal requirement between events in E, and can be viewed as a special type of episode without state constraint. A simple temporal constraint(Dechter, Meiri, and Pearl 1991) is a tuple $< e_S, e_E, lb, ub >$ where:
  - e_S and e_E in E are the start and end events of the temporal constraint.
  - lb and ub represent the lower and upper bounds of the duration between events e_S and e_E , where $lb \leq Time(e_E) - Time(e_S) \leq ub$, $(lb \in \mathbb{R} \cup -\infty, ub \in \mathbb{R} \cup +\infty)$.

Activities and composable behaviors

The *domain* of the planning problem is given by the *durative activities* and the *behaviors*.

Durative activities are similar to the ones defined in the temporal planning literature and have a bounded controllable duration, and a set of discrete effects and conditions defined *at start*, *over all* and *at end*. The conditions can also be continuous.

In Scotty we define a new type of activities that we call **behaviors**. These behaviors preserve the main elements of the durative activities but differ from these in that they also encode continuous non-linear effects that can represent, for example, the dynamics of a robot moving in an environment. We will consider that *durative activities* are a special case of *behaviors* with no continuous effects.

Consider for example a behavior *collect-water-column-data* of an autonomous underwater vehicle. This behavior represents the AUV taking measurements in a water column, and has the discrete effect *at end* of having collected the data (*data-collected*). This behavior also has the discrete *over all* conditions that the engine needs to be on and the sensors activated while the behavior is being executed. It also has continuous *over all* conditions that specify that the AUV needs to stay within some coordinates while taking the measurements $(x, y \in Region)$. Finally, the depth before collecting the data needs to be within 100 and 120 meters $(100 \leq z \leq 120)$, and the depth at the end has to be smaller than 40 meters $(z \leq 40)$. These represent the continuous conditions *at start* and *at end*.

Moreover, the AUV needs to move in an ascending spiral of fixed radius while collecting the data. This is represented by the non-linear dynamics of the behavior:

$$\dot{x}(t) = v \cdot cos(\theta) \tag{1}$$
$$\dot{y}(t) = v \cdot sin(\theta) \tag{2}$$
$$\dot{z}(t) = v_z(t) \tag{3}$$
$$\dot{\theta}(t) = \omega \tag{4}$$
$$\frac{v}{\omega} = R \tag{5}$$

The dynamics of this behavior affect state variables x, y, z and θ and their velocities. These dynamics are driven by control variables v_x, v and ω which are assumed to be controllable within some bounds (actuation limits) during execution. Additionally, there may be other state constraints affecting the execution of this behavior. These constraints can be user-specified as part of the input QSP ('ensure the battery level is always greater than 25%' or 'stay within the high reception region' for example), or imposed by other activities or behaviors being executed at the same time.

An important property of behaviors is that they are **composable**. That is, behaviors can be pieced together sequentially one after another. Behaviors have **inlet** and **outlet con-**

nectors. In this example both the inlet and outlet of the behavior are of type *pose*. That is, behavior *collect-water-column-data* takes a starting pose and *transforms* it into a different pose at the end of its execution. Behaviors with compatible connectors can be connected together. For example, in order to start the *collect-water-column-data* behavior, the AUV first needs to reach the water column region at a certain depth. A prior behavior *navigate*, also with *pose* connectors, could be the one taking the AUV from it's starting pose to a valid starting pose for *collect-water-column-data*. In the same way, *collect-water-column-data* could be followed by an additional *navigate* behavior that could take the AUV to its final rendezvous region. A third behavior *surface* could have a *position* inlet requiring the AUV to be in the rendezvous region before ascending to the surface. Because the *position* connector (x, y, z) is less specific than the *pose* connector (*position* and *orientation*), the *navigate* behavior can connect to the *surface* behavior. Scotty ensures that the connector constraints are satisfied in the solution plan. While multiple durative activities can be executed simultaneously, we only allow simultaneous execution of behaviors that do not affect the same state variables.

Solution

Scotty returns a plan satisfying the temporally-extended goals and conditions described in the input QSP. The solution plan consists of a list of scheduled behaviors where each behavior b has an execution start time t_b, duration d_b, and a trajectory of the value of each of its control variables from t_b to $t_b + d_b$. The returned plan also needs to satisfy all the behavior discrete and continuous conditions as well as the connection constraints between behaviors.

Work so far

In the initial stage of this thesis we first approached the problem of combining heuristic forward search symbolic planning with trajectory optimization by solving a simplified problem. Instead of considering arbitrary dynamics, in this stage continuous effects are assumed to be linear time varying. While we allow full discrete activity planning, we restrict the continuous effects to the ones described before, and we assume the environment is obstacle free. Therefore we do not use the robot behaviors defined earlier yet, but durative activities with some modifications.

We solve this problem by combining heuristic forward search and trajectory optimization through solving linear programs that are used to resolve the continuous effects and to test the consistency of partial plans.

This stage was completed in Spring 2015 and resulted in an IJCAI-15 publication (Fernandez, Karpas, and Williams 2015).

Simplified Problem Statement

The simplified problem statement is framed as a standard PDDL 2.1(Fox and Long 2011) planning problem with some modifications that allow us to define activities with continuous effects that depend on *bounded control variables*. These bounded control variables represent, in general, velocities

```
(:durative-action navigate
:control-variables ((velX) (velY))
:duration (and (<= ?duration 5000))
:condition (and
                (over all (>= (velX) -4)) (over all (<= (velX) 4))
                (over all (>= (velY) -4)) (over all (<= (velY) 4))
                (over all (<= (x) 700)) (over all (>= (x) 0))
                (over all (<= (y) 700)) (over all (>= (y) 0))
                (at start (AUV-ready)))
:effect (and
           (at start (not (AUV-ready)))
           (at end (AUV-ready))
           (increase (x) (* 1.0 (velX) #t))
           (increase (y) (* 1.0 (velY) #t))))
```

Figure 1: Navigate activity modified by continuous control variables *velX* and *velY*.

that are bounded (the actuation limits of the system). Similarly to PDDL 2.1, durative activities have a bounded controllable duration, discrete effects and continuous and discrete conditions defined *at start*, *over all* and *at end*. Continuous conditions are limited to linear inequalities over the state variables according to:

$$\mathbf{c^T x'} \leq 0 \tag{6}$$

, where $\mathbf{x'} = (x_1, \ldots, x_{n_x}, 1)^T$ and $\mathbf{c} \in \mathbb{R}^{n_x+1}$ is a vector of coefficients, with n_x being the number of state variables of the system.

The simplified problem statement differs from PDDL 2.1 in the effects on continuous variables. Each activity has a set of control variables, which can be seen as continuous parameters — each of those constrained by lower and upper bounds. The continuous effects of the activity are similar to those of PDDL 2.1, except they are affected by the value chosen for the control variables. Here we restrict each continuous effect to involve only a *single* control variable, c_{var}, and thus each continuous effect can be defined by $\langle x, c_{var}, k \rangle$, where x is a state variable, c_{var} is a control variable, and k is a constant.

In the simple case, where a single continuous effect $\langle x, c_{var}, k \rangle$ is active from time t_{start} to time t_{end} with c_{var} fixed to a constant value of c throughout the duration, then $x(t)$, the value of state variable x at time t is defined by $x(t) = x(t_{start}) + k \cdot c \cdot (t - t_{start})$ with $t_{start} \leq t \leq t_{end}$.

Multiple continuous effects on the same state variable are additive, and thus $x(t)$ is defined by:

$$x(t) = x(0) + \int_0^t C_x(\tau) d\tau \tag{7}$$

where $C_x(t)$ is the sum of the values of the control variables in active continuous effects modifying x at time t (represented by the set E).

$$C_x(t) = \sum_{\langle x, c_{var}, k \rangle \in E} k \cdot c_{var}(t) \tag{8}$$

where $c_{var}(t)$ denotes the value chosen for the control variable c_{var} at time t. An example *navigate* activity for a robot is shown in Figure 1. Note the bounded *control variables* velX and velY.

In this stage the *goal* simply consists of the discrete and continuous conditions that need to hold at the end. Temporally-extended goals are not supported yet.

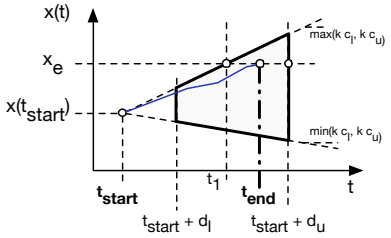


Figure 2: Flow tube with its reachable region (shaded area). The solid blue line represents an example valid state trajectory. The flow tube contains all valid state trajectories.

Approach

In order to solve this planning problem, we merge the powerful representation of continuous effects based on *flow tubes* from Kongming (Li and Williams 2008; Li 2010) with the efficient solving method based on heuristic forward search and linear programs for consistency checking that COLIN (Coles et al. 2009) uses.

Each continuous effect of an activity is represented by a flow tube. Flow tubes represent the reachable state space region, that is, the values that the state variable can take after the activity is started. Remember that here we restrict continuous effects to linear time varying effects.

If no other flow tubes affect state variable x between t_{start} and t_{end}, then the reachable region of x represented by the flow tube $f(d_l, d_u, c_{var}, x, k)$ of an activity executed between t_{start} and t_{end} is defined by the following equations:

$$x(t_{end}) = x(t_{start}) + k \cdot \int_{t_{start}}^{t_{end}} c_{var}(t) \cdot dt \quad (9)$$

with
$$c_l \leq c_{var}(t) \leq c_u \quad (10)$$
$$d_l \leq t_{end} - t_{start} \leq d_u \quad (11)$$

where $x(t_{start})$ is the value of the state variable before the activity is executed. Note that, if the value of the control variable c_{var} is constant during the execution of the activity, equation 9 reduces to $x(t_{end}) = x(t_{start}) + k \cdot c_{var} \cdot (t_{end} - t_{start})$.

Figure 2 shows a flow tube. Note that any point in the shaded region (reachable region) can be reached at the end of the activity by carefully choosing the appropriate activity duration and control variable value. In the figure, we can see how the state value x_e can be reached as fast as in $t_{end} = t_1$ if the control variable c_{var} is constant and takes its maximum possible value (c_u), or as late as $t_{end} = t_{start} + d_u$ if $c_{var}(t)$ takes smaller values.

An important characteristic of flow tubes, is that they provide a compact encoding of all feasible trajectories. In order to find a feasible plan, we connect flow tubes forward using heuristic forward search, as explained next.

We use a method based on heuristic forward search and linear programs for consistency checking that is borrowed from COLIN. The main difference is that in our formulation, continuous effects support control variables and are represented by flow tubes and, therefore, the state evolution constraints are different from COLIN's.

Activities are represented by their start and end events, analogous to the start and end snap actions used by many temporal planners (Long and Fox 2003; Coles et al. 2008). The planner needs to find the ordered sequence of start and end events that takes the system from the initial conditions to the goal and the execution time of each event. We also need to find a trajectory for the values of each activity control variable between the start and end events of the activity ($c_{var}(t)$ with $t_{start} \leq t \leq t_{end}$).

As explained before, we use heuristic forward search to find the sequence of start and end events that form a fixed plan, and linear programs to check the consistency of partial plans. The search uses Enforced Hill Climbing, which has proven to be effective in this type of problems (Hoffmann and Nebel 2001).

Search states contain the set of propositions that are known to be true due to discrete effects, and are augmented with the ongoing activities list and the bounds for all state variables. The ongoing activities list keeps track of the activities that have started but not finished at that state and is needed to keep track of the active overall discrete and continuous constraints. The lower and upper bounds for the state variables are used to prune sections of the search tree that are necessarily not feasible.

Each search state defines a partial plan as the current sequence of start and end events, and is tested for consistency with a linear program. The partial plan is feasible if the linear program has a solution. In this linear program the decision variables are the event execution times and the values of the state variables at each event. The constraints include activity duration, start, end and overall conditions and state evolution constraints that are built from the current sequence of events. These constraints are the same ones that COLIN uses (Coles et al. 2012) except for the state evolution constraint, however, due to the presence of control variables. This constraint is given by the flow tube reachability equation 9. Because the values of the control variables can change during the activity execution, and the start and end times of the activity are variables of the linear program, this equation is not linear if control variables are decision variables of the linear program. However, we can redefine the reachability region of the flow tube with the following linear inequalities:

$$x_{end} \geq x_{start} + \min(k \cdot c_l, k \cdot c_u) \cdot (t_{end} - t_{start}) \quad (12)$$
$$x_{end} \leq x_{start} + \max(k \cdot c_l, k \cdot c_u) \cdot (t_{end} - t_{start}) \quad (13)$$

, where c_l and c_u are the bounds of the control variable. Note that $\min(k \cdot c_l, k \cdot c_u)$ represents the minimum rate of change of $k \cdot c_{var}$ and reduces to $k \cdot c_l$ when $k > 0$. The more complicated expression is needed to preserve generality when $k < 0$. The same applies to the maximum rate of change. These linear inequalities represent the same flow tube reachability region described by equation (9) if each of the activity's control variables appear in only one continuous effect.

The consistency program is solved for each state in the search tree to determine the feasibility of the partial plan, and to extract the event times ($t_1 \ldots t_6$), state variable values

($x_1 \dots x_6$), and control variables. These values keep changing as more steps are added to the plan during search. In order to find the state variable lower and upper bounds, the LP is solved twice per state variable (to minimize and maximize its value).

If the current search state is determined to be consistent, its heuristic value is computed and the state is added to the queue. If the state satisfies the goal conditions, a valid fixed plan has been found and Scotty proceeds to extract the flexible plan next. The last linear program used to extract the fixed plan tries to minimize the makespan of the plan, although a different optimization objective could be chosen.

The **heuristic** function used by Scotty is essentially the same used by COLIN, with minor modifications due to the use of control variables. The heuristic value for a state is the number of start or end events to reach the goal in the relaxed plan. The planning graph that COLIN expands keeps track of the state variables lower and upper bounds for each fact layer, with the caveat that activities can only grow these bounds, in a similar fashion to Metric-FF (Hoffmann 2003). COLIN calculates the positive gradient affecting each state variable by adding the positive rates of change of each ongoing activity (similarly with the negative gradient). In Scotty's case, these positive and negative gradients are found by adding the maximum (and minimum) rates of change given by the bounds of the control variables affecting each activity.

Future research

Currently, I am working on moving from the limited linear action model to a more general non-linear model that supports more interesting robot behaviors.

In particular, I am working towards supporting the following:

1. Advanced robot behaviors with non-linear dynamics
2. Temporally-extended goals and user-specified global constraints
3. Search heuristics through behavior approximations

Now instead of planning with durative activities, we plan with **behaviors**. Behaviors still support the standard discrete conditions and effects, but are extended to allow non-linear dynamics and more expressive user-specified constraints. Also, as described in the problem statement, behaviors are *composable* elements that can be connected sequentially and transform **inlets** to **outlets**. Behaviors that produce an outlet of a certain kind can connect to a behavior accepting the same kind of inlet.

As an example, imagine that the dynamics of a *navigate* behavior are given by the following Dubin's equations:

$$\dot{x}(t) = v \cdot cos(\theta) \tag{14}$$
$$\dot{y}(t) = v \cdot sin(\theta) \tag{15}$$
$$\dot{\theta}(t) = u(t) \tag{16}$$
$$|u(t)| \leq \frac{1}{\rho} \tag{17}$$

, where v is the velocity, that can be fixed or not, and ρ is the minimum turning radius.

Note that now we not only deal with much more complex dynamics than before but we also need to handle **switch points**: the connection states between subsequent behaviors. These were trivial to handle in simplified case with the linear program formulation, as they corresponded directly with the decision variables of the LP (the state variables at the switch points). However, this is harder to handle in this case. The way we deal with this is by using the connector (**inlets** and **outlets**) specification of behaviors. In effect, the navigate activity has a *pose* ($\langle x, y, \theta \rangle$) inlet and a pose outlet, meaning that it takes the robot from a starting pose to an ending pose. When connecting the *navigate* behavior with a *collect-data* behavior with analogous specification, the ending pose after *navigate* becomes the start pose for *collect-data*. Another *deliver* behavior may have an inlet of type *position* ($\langle x, y \rangle$), a subset of the *pose* type. Because the outlet of behavior *navigate* is a *pose* which is more specific than a *position* inlet, both behaviors can connect (*navigate* finishes at some position and orientation, but *deliver* only enforces the position to be inside the delivery region). We handle these relations by framing a joint *trajectory optimization* program combining sequential behaviors in which the connection constraints between them are added explicitly. Although one may think that framing large optimization programs like this is not practical, some combined task and motion planners have followed this approach with good results (Toussaint 2015). As Toussaint notes in his paper, this approach has the great advantage that it does not require discretizing the continuous states in advance, and that it lets efficient solvers choose the best switch poses, which would otherwise require very fine discretization or many randomly sampled poses with many combinatorial possibilities if done outside the trajectory optimization paradigm.

We now also consider global temporally extended goals and user-specified constraints. This may be things such as completing one of the objectives of a mission within a given time or ensuring that the robot never leaves a certain safe region. This is represented by a *Qualitative State Plan* (QSP). The nodes of the graph represent *events*, and the black arcs in between them, *episodes*. *Episodes* denote conditions that need to be satisfied between events. The temporal relations between events are specified with simple temporal constraints. These specifications will be modeled as additional constraints in the global trajectory optimization program and will affect the selection of the switch points.

Finally, recall that the heuristic that we used for the simplified problem was based on the *relaxed planning graph*. Unfortunately, this approach cannot be easily adapted to complicated non-linear behaviors. In effect, computing the bounds of the state variables after executing some non-linear behavior is not only computationally expensive, but also, it cannot be determined in general if any arbitrary value inside those bounds can be reached (this was the case in the linear effects case).

In this thesis we will try to optimize the forward search by pruning options that cannot be feasible in order to avoid wasting time on them. We will do so by using approxima-

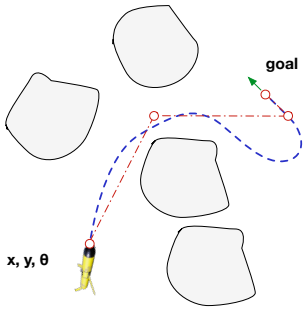


Figure 3: An outer approximation for the Dubin's smooth path given by an infinite curvature (straight-line) path

tions to the non-linear behaviors that are faster to compute. We want to use outer approximations that guarantee that if no solution can be found using the approximation, no solution exists for the more detailed model either. However, some returned solutions using the approximated model may not be feasible when checked with the more detailed model.

Figure 3 shows an example of an approximation for the Dubin's dynamics model that the *navigate* activity follows. The smooth path resulting from Dubin's equations (shown in blue) can be approximated by a piece-wise straight-line path (in red). The linear approximation is faster to compute and it can be shown that if no linear path exists from the starting conditions to the goal conditions, no Dubin's path exists either. On the other hand, there could exist an obstacle-free linear path between the start and the goal poses but no Dubin's equivalent path (curvature constraints may be violated, for example). We can use this approximation to calculate lower and upper bounds on duration (using minimum and maximum velocities) and to prune search states leading to infeasible outcomes.

References

Cambon, S.; Alami, R.; and Gravot, F. 2009. A Hybrid Approach to Intricate Motion, Manipulation and Task Planning. *The International Journal of Robotics Research* 28(1):104–126.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with Problems Requiring Temporal Coordination. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, 892–897.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2009. Temporal Planning in Domains with Linear Processes. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 1671–1676.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 42–49.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2012. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research (JAIR)* 44:1–96.

Dai, H.; Valenzuela, A.; and Tedrake, R. 2014. Whole-body motion planning with centroidal dynamics and full kinematics. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, 295–302. IEEE.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence*.

Fallon, M.; Kuindersma, S.; Karumanchi, S.; and Tedrake, R. 2015. An Architecture for Online Affordance-based Perception and Whole-body Planning. *Journal of Field ...*.

Fernandez, E.; Karpas, E.; and Williams, B. C. 2015. Mixed Discrete-Continuous Heuristic Generative Planning Based on Flow Tubes. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 1565–1572.

Fox, M., and Long, D. 2011. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *CoRR* abs/1106.4561.

Garrett, C. R.; Lozano-Pérez, T.; and Kaelbling, L. P. 2014. FFRob: An efficient heuristic for task and motion planning. *lis.csail.mit.edu*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 253–302.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *J Artif Intell Res(JAIR)* 20:291–341.

Hofmann, A., and Williams, B. C. 2006. Robust Execution of Temporally Flexible Plans for Bipedal Walking Devices. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6-10, 2006*, 386–389.

Léauté, T., and Williams, B. C. 2005. Coordinating Agile Systems through the Model-based Execution of Temporal Plans. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, 114–120.

Li, H. X., and Williams, B. C. 2008. Generative Planning for Hybrid Systems Based on Flow Tubes. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, 206–213.

Li, H., and Williams, B. C. 2011. Hybrid Planning with Temporally Extended Goals for Sustainable Ocean Observing. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*.

Li, H. X. 2010. *Kongming: a generative planner for hybrid systems with temporally extended goals*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Long, D., and Fox, M. 2003. Exploiting a Graphplan Framework in Temporal Planning. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*, 52–61.

Srivastava, S.; Fang, E.; Riano, L.; and Chitnis, R. 2014. Combined task and motion planning through an extensible planner-independent interface layer. ... *(ICRA)*.

Toussaint, M. 2015. Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 1930–1936.

Planning with Flexible Timelines in the Real World

Alessandro Umbrico, Marta Cialdea Mayer
University Roma TRE, Italy
alessandro.umbrico@uniroma3.it

Andrea Orlandini
CNR - National Research Council of Italy
Institute of Cognitive Science and Technology

Abstract

Planning is a core field of Artificial Intelligence since its beginnings. Broadly speaking, planning techniques aim at providing artificial agents with the capability to autonomously solve "complex" problems. Several planning techniques have been introduced in the literature that employ different approaches for modeling and solving planning problems.

My PhD research activities concern timeline-based approach to planning. Timeline-based planning is a particular Temporal Planning paradigm which has been successfully applied to solve real-world problems. Despite it practical success there is not a shared view of this planning approach. There are several timeline-based frameworks that have been introduced in the literature each of which applies its own *interpretation* of timelines, timeline-based plans and planning domains.

In this regard the objective of my PhD is to analyze the features of the different existing timeline-based systems and to provide a complete characterization of timeline-based approach by providing a semantics for the related planning concepts, defining a methodology to model domains and problems and by defining domain independent heuristics and develop a suited timeline-based planning framework, called EPSL.

Introduction

Timeline-based planning has been introduced in early 90s (Muscettola 1994), it takes inspiration from the classical control theory. It models a complex system by identifying a set of relevant features that must be controlled over time. This approach has been successfully applied in several real-world contexts (especially in space applications) thanks to take into account the temporal aspects of the problem. Several planning frameworks have been developed for the synthesis of timeline-based P&S applications, e.g. EUROPA (Barreiro et al. 2012), ASPEN (Chien et al. 2010), APSI-TRF (Cesta et al. 2009) or IXTET (Laborie and Ghallab 1995).

Despite its practical success, there is a lack of formalization of timeline-based planning related concepts. There is not a uniform and shared view of concepts like timelines, timeline-based plans, domains and problems. Every framework applies its own *interpretation* of timeline-based planning. This results in different ways of considering timeline-based problems and also, different ways of modeling and solving such problems.

In this context, my PhD research goal is to characterize timeline-based planning approach from different point of views and develop a suited planning framework, called EPSL - the Extensible Planning and Scheduling Library. Namely, we aim at understanding timeline-based planning by providing an acceptable semantics for the related planning concepts, defining a methodology to model and solve problems by means of timelines.

The following sections introduce timeline-based planning approach by exploiting the formalization we have proposed in some recent works (Cialdea Mayer, Orlandini, and Umbrico 2015; 2014) and the EPSL planning framework that I'm currently developing. Later sections describe an interesting application of EPSL (and the timeline-based approach) to a manufacturing real-world context for the development of a knowledge-based control module (KBCL - the Knowledge-Based Control Loop). In particular, this application gave an important contribution for the definition of a hierarchical modeling approach for timeline-based planning and a domain independent heuristic that we have implemented and tested in the EPSL framework.

Finally we briefly present some ongoing works on the comparison of our "vision" of timeline-based planning and EPSL with EUROPA which is one of the most known timeline-based software environment in the literature.

Timeline-based Planning Approach

The main result concerning the formalization of timeline-based planning approach is represented by our work (Cialdea Mayer, Orlandini, and Umbrico 2015). The key contribution of this work, which builds and extends previous works (Cialdea Mayer, Orlandini, and Umbrico 2014), is to provide a formal stand-alone definition of the main concepts of timeline-based planning and the relative *controllability properties*, independently from the concrete structure exploited to represent timelines.

The importance of this feature is due to the fact that representing a flexible timeline-based plan as a *temporal network* entails a sort of simplification of the associated plan structure, thus causing a loss of information on the causal/temporal "dependencies" among its components. Indeed, such information can be useful for planning engines

(for instance to define suited heuristics as we'll see in the next sections) and in general, for supporting a more detailed analysis of the relevant features enclosing in the generated plans.

The timeline-based approach pursues the idea that planning and scheduling for controlling complex physical systems consists of the synthesis of desired temporal behaviors (i.e. *timelines*). Thus, a *planning domain* is modeled as a set of features with an associated set of temporal functions on a finite set of values.

The time-varying features of the domain can be modeled by means of *multi-valued state variables*. The possible evolutions of these features are described by some causal laws and limited by domain constraints. These are specified in a *domain specification*. A timeline-based planner must find a sequence of decisions that brings the timelines into a final desired set, satisfying the domain specification and goals. Thus, a domain specification must provide the set of causal and temporal constraints that specify which value transitions are allowed for the state variables, and the minimal and maximal duration of each valued interval.

State variables A state variable models a particular feature of the domain that must be controlled over time. Formally it is characterized by four components: the set V of values representing the possible state or actions the feature can assume or perform over time; a function T mapping each value $v \in V$ to the set of values that are allowed to follow v; a function γ tagging each value $v \in V$ with information about its controllability; a function D setting upper and lower bounds on the duration of each value of the variable. In particular, the *controllability tagging fucntion* γ tags each value $v \in V$ as controllable $\gamma(v) = c$ or not $\gamma(v) = u$. If a value v is *controllable* it means that the planner (or the executor of the plan) can decide the actual duration of the value. If a value v is *uncontrollable*, instead, the planner can decide its start time but the planner cannot decide its end time. Namely, the planner cannot make any hypothesis about the actual duration of uncontrollable values during the solving process.

Synchronization rules A domain specification must provide global constraints that coordinate the temporal behaviors of the state variables. Such relations are specified by means of *synchronization rules* that constrain values of different state variables. Namely, synchronization rules specify how the domain features must behave in order to perform some complex tasks.

Formally, a synchronization rule is an expression of the form:

$$a_0[x_0 = v_0] \rightarrow \exists\, a_1[x_1 = v_1] \ldots a_n[x_n = v_n]\, .\mathcal{C}$$

where (i) $a_0, \ldots, a_n$ are distinct *token variables*; (ii) for all $i = 0, \ldots, n$, x_i is a state variable and $v_i \in values(x_i)$; and (iii) $\mathcal{C}$ is a positive boolean formula where only the token variables $a_0, \ldots, a_n$ occur.

Token variables represents particular instances of values of the domain state variables. It is important to point out that the use of token variables allows to specify multiple instances of the same value of a state variable in the right-hand part of a synchronization rule. The left-hand part of the synchronization rule, $a_0[x_0 = v_0]$, is called the *trigger* of the rule and represents the value the rule can be applied to.

Timelines A timeline represents the temporal evolution of a system component up to a given time (the *horizon*). It is made up of a sequence of valued intervals, called *tokens*, each of which represents a time slot in which the variable assumes a given value. It is important to point out that, when planning with timelines, time flexibility is taken into account by allowing token durations to range within given bounds. A token for a variable $x = (V, T, \gamma, D)$ is completely described by representing its start and end "times" with temporal intervals as follows:

$$x^i = (v, [e, e'], [d, d'], \gamma(v))$$

Thus a timeline FTL_x is a for a state variable $x = (V, T, \gamma, D)$ is a finite sequence of tokens of the form:

$$FTL_x = x^1 = (v_1, [e_1, e'_1], [d_1, d'_1], \gamma(v_1)), ..., x^k = (v_k, [e_k, e'_k], [d_k, d'_k], \gamma v_k)$$

It is important to point out that once a token x^i is embedded in a timeline, the time interval to which its start points belongs can be easily computed by considering the end time of the previous token, $start\_time(x^{i+1}) = end\_time(x^i)$ (where the start time of the first token x^0 is $[0, 0]$).

A *scheduled timeline* is a particular case where each token has a fixed end time $[t, t]$. A *schedule* of a timeline FTL_x is essentially obtained from FTL_x by narrowing down to singleton (i.e. time points) the end times of the tokens. The schedule of a token corresponds to one of the valued intervals it represents which is obtained by choosing an exact end point in the allowed interval without changing its duration bounds. In this regards, a scheduled timeline is a sequence of scheduled tokens that satisfy their duration bounds.

Flexible plans The main component of a flexible plan is a set of timelines representing different sets of scheduled ones. It may be the case that not every scheduled timelines satisfy the synchronization rules of the domain. In order to guarantee that every set of scheduled timelines represented by a given flexible plan π (i.e. the different ways of executing π) is valid with respect to the underlying planning domain, the plan has to be equipped with additional information about the temporal relations that have to hold in order to satisfy the synchronization rules of the domain. Namely, the representation of a flexible plan must also include information about the relations that must hold between tokens in order to satisfy the synchronization rules of the planning domain.

In general, a flexible plan includes a set of temporal constraints ($\mathcal{R}$) on tokens $\pi = (\mathbf{FTL}, \mathcal{R})$. When there are different ways to satisfy a synchronization rule by the same set **FTL** of flexible timelines, there are also different (valid) flexible plans with the same set of timelines **FTL**; each of them represents a different way to satisfy synchronizations.

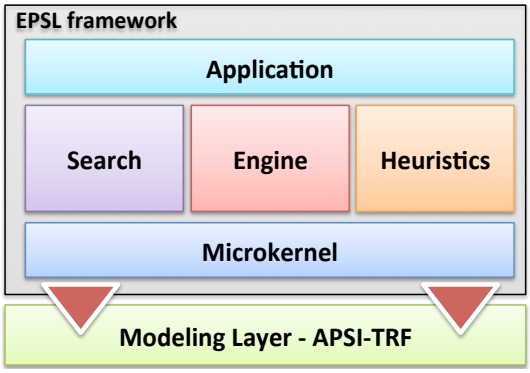


Figure 1: EPSL architecture

The Extensible Planning and Scheduling Library

Timeline-based applications, typically, are strictly related to the specific domain they have been designed for. It is hard to exploit "past experience" in order to adapt already developed applications to different context with different features. Thus, it is usually start developing new applications from scratch.

In this regard, the EPSL (the Extensible Planning and Scheduling Library) (Umbrico, Orlandini, and Cialdea Mayer 2015) is a layered software framework (built on top of APSI-TRF (Cesta and Fratini 2008)) which aims at defining a flexible software environment for supporting the design and development of timeline-based applications. The key point of EPSL is its interpretation of a planner as a *modular* solver which combines together several elements to carry out its solving process.

Figure 1 shows the main architectural elements of the EPSL architecture. In particular, the *Engine Layer* is the architectural element responsible for managing the portfolio of algorithms (called *resolves*) a planner can use to actually solve timeline-based problems. The higher is the number of available resolver the higher is the solving capability of the framework. Indeed, the solving process of an EPSL-based planner consists of a plan refinement procedure which iteratively refines the plan by solving "undesired" conditions, called *flaws*. A flaw represents a particular condition which threats the completion or the consistency of the current plan (e.g. a planning goal). Every resolver is responsible for detecting and solving a particular type of flaw. The set of available resolvers determines what an EPSL-based planner can actually do to solve a problem. Similarly, the *Heuristic Layer* is the architectural element responsible for managing the set of available criteria an EPSL-based planner can use to select flaws when solving.

Thus, the EPSL solving approach can be "easily" adapted to the particular problem to address by changing the set of resolvers and heuristics the planner can use (i.e. the planner configuration). As a matter of fact, the particular strategy or heuristics applied can strongly affect the behavior and the performance of a planner.

Knowledge-Based Control Loop

The KBCL is a knowledge-based control module developed within the GECKO project I have collaborated to during my PhD. The reader may refer to several works (Stefano et al. to appear; Borgo et al. 2015; 2014a; Carpanzano et al. 2015; Borgo et al. 2014b) for a detailed description of the approach and the specific application context.

The key direction of the GECKO project has been to endow the control architecture of an agent with a knowledge reasoning mechanism capable of representing the actual capabilities of the related agent. Thus we made a tight integration between knowledge representation techniques with and timeline-based planning by exploiting the EPSL framework.

In general, a plan-based controller can endow an agent with the deliberative capabilities needed to autonomously perform complex tasks. In particular contexts like the RTS (Reconfigurable Transportation System) of the manufacturing case study we have considered within the GECKO project, the dynamic nature of the system we want to control does not guarantee a continuous control process capable to face all the particular situations/configurations.

Indeed, the specific capabilities of a Transportation Module (TM) of the RTS can be affected by many factors, e.g. a partial failure of the internal elements or a reconfiguration of an RTS plant (see cited works for further details). Thus it is not always possible to to design a plan-based controller which is able to efficiently handle all these situations. The higher is the complexity of the planning domain the higher is the time needed to synthesize the plans.

Our proposed solution was to "extend" the control loop of an agent with a knowledge-reasoning mechanism capable of representing its actual capabilities. In this way, it is possible to simplify the planning model specification by considering only the actual capabilities of the agent (e.g. a TM of the RTS) to control. In particular, the knowledge reasoning mechanism allows to realize a continuous control process by dynamically synthesizing the timeline-based planning model every time a change in the capabilities of the agent (i.e. the TM) occurs.

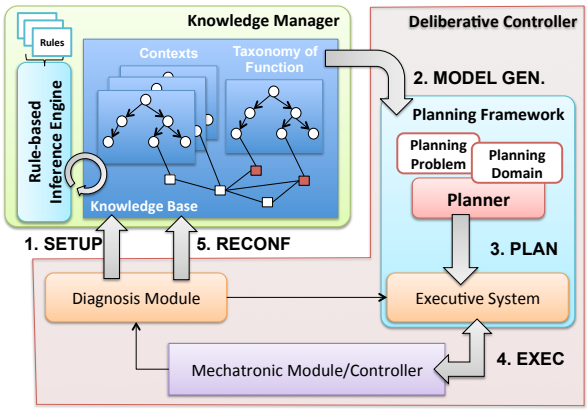


Figure 2: The KBCL module architecture

Figure 2 shows the extended control loop which results

from the integration of two ”big boxes”. The *Knowledge Manager* which contains the knowledge about the agent to control, and the *Deliberative Controller* which represents the ”classical” plan-based control architecture where EPSL is integrated in an execution environment for the synthesis and execution of timeline-based plans.

The ontological approach The Knowledge Manager of Figure 2 relies on a suited ontology which captures the general knowledge of the manufacturing environment. The ontology contains (i) a context-based classification of the information about the agent and production environment, and (ii) a taxonomy of function which classifies the set of functions a generic agent can perform in a manufacturing environment.

The Knowledge Manager exploits the ontology to build and manage the Knowledge Base (KB) of the specific agent (i.e. TM) to control. In particular the context-based approach classifies information according to three contexts that characterize the agent from three different point of views. The *internal context* characterizes the internal structure of the agent and its components. The *local context* characterizes environment in terms of other agents or elements of the production environment the agent must interact with. The *global context* contains information of interest for all the agent composing the shop floor, e.g. the type of product to work or information about the performance of the factory.

The KBCL in action The management of the KB, the generation of the planning domain and the continous monitoring of the information concerning the actual status of the agent (and its environment) are complex activities that must be properly managed by the KBCL process at runtime. In this regard the KBCL is composed by the following phases (depicted in Figure 2: (i) the *setup* phase; (ii) the *model generation* phase; (iii) the *plan and execution* phase; (iv) the *reconfiguration* phase.

Broadly speaking, the *setup phase* generates the KB of the agent by processing the raw data received by the agent (a TM in the GECKO project) through the *Diagnosis Module* of Figure 2. The resulting KB completely describes the agent to control in terms of its structure, its capabilities and the related production environment. The *model generation* phase exploits the KB of the agent to automatically generate the timeline-based planning domain needed by the *Deliberative Controller* to actually control the device.

When the planning model has been generated the *plan and execution* phase starts. The KBCL process behaves like a classical plan-based controller during this phase. However, whenever a structural changes occurs in the agent or its environment e.g. a failure of an internal component or a failure of a collaboratore of the shop floor, the *reconfiguration* phase starts. The *reconfiguration* phase determines a new iteration of the KBCL cycle and a new version of the KB and a new version of the timeline-based planning model are generated.

A Hierarchy-based Modeling Appraoch

When applying timeline-based approach usually, we must control an ”artificial agent” able to perform some complex tasks in a specific working environment e.g. a TM of the GECKO project. In order to provide a suited timeline-based model it is necessary to capture all the features, the operational and temporal constraints that characterize a specific domain. In this regards, exploiting the context-base analysis described in the previous section to characterize the knowledge about the functional capabilities of an agent, the modeling approach we propose, follows a functional decomposition of the domain by identifying three relevant types of state variables. They are (i) the *functional variables*, (ii) *primitive variables* and (iii) *external variables* (see (Umbrico, Orlandini, and Cialdea Mayer 2015) for further details).

A *functional variable* provides a logical representation of the agent in terms of the high-level task the agent can perform, notwithstanding its internal composition. A *primitive variable* models a specific physical/logical component of the system. Values of such a variable correspond to concrete state/actions the related element is actually able to assume/perform over time. Finally, an *external variable* provides a logical view of an element whose behavior is not under the control of the system but affect the execution of its functionalities. Such a variable models conditions that must hold in order to successfully perform internal activities.

Synchronization rules specify constraints between different variables of the planning domain. These rules allow to further constrain the behaviors of the domain state variables in order to safely realize complex tasks. In this regard, given the described modeling approach, synchronization rules can be used to specify how the high-level functionalities (i.e. the values of the functional state variables) are implemented by the agent. A synchronization rule specifies the set of *primitive* and/or *external* values and the needed temporal relations that allow the agent to successfully perform the related high-level function (i.e. the *functional* value the synchronization applies to).

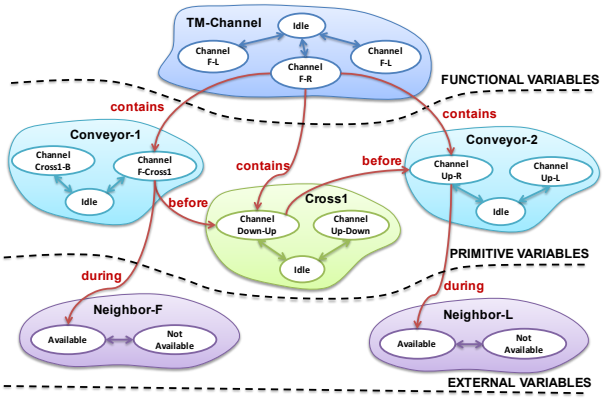


Figure 3: A planning domain example

Figure 3 shows an example of a planning domain obtained by applying the modeling approach described to a Transportation Module (TM) of the manufacturing plant in the

GECKO project. The functional state variable *TM-Channel* models the transportation tasks the module is able to perform (e.g. *Channel-F-B*, *Channel-R-L*). The primitive state variables model the internal component of the TM in terms of the operations they can perform (e.g. a conveyor of the module is able to move a pallet between two internal position of the module). The external state variables model the status of other agents (i.e. other TMs) of the plant the TM must cooperate with.

Finally the read arrows of Figure 3 models constraints among values of the state variables that must be satisfied to perform the tasks. Specifically, the constraints describe the way the TM can implement the possible transportation tasks. E.g. the TM performs a "Channel-F-R" by moving the pallet from position "F" to position "Cross1" by means of "Conveyor-1". Then the "Cross1" moves the pallet from position "down" to position "up". Finally, the "Conveyor-2" moves the pallet from position "up" to position "L".

A domain-independent heuristic It is possible to observe that a *synchronization rule* basically, represent a *dependency* between two or more variables and their timelines. Thus, given a timeline A and a timeline B, a synchronization rule $S_{A,B}$ from a token $x \in A$ to a token $y \in B$ implies a dependency between these timelines. Namely, tokens on timeline B are subject to tokens on timeline A.

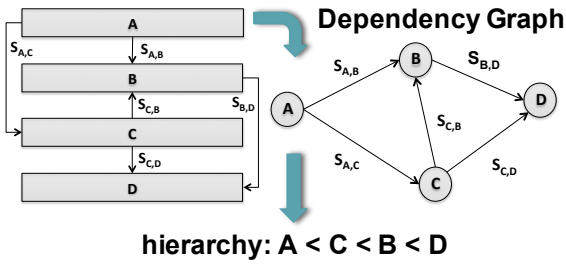


Figure 4: Extracting hierarchy from synchronizations

Given this assumption, it is possible to build a *dependency graph* (DG) among timelines by analyzing synchronization rules. Figure 4 shows a set of timelines with synchronization rules and the resulting dependency graph. The nodes of the graph represent timelines (or state variables) and edges represent dependecies between them (i.e. at least a synchronization rule between tokens of the related timelines exists).

Given the DG, it is possible to extract the hierarchy of the domain. An edge from a node A to a node B in the DG represents a dependency between timeline A and timeline B. Consequently, the hierarchy level of timeline A is not lower than the hierarchy level of timeline B. If not path in the DG connects B to A (i.e. no cycle is detected) then the hierarchy level of A is higher than the hierarchy level of B. Thus if the DG contains a cycle among two or more nodes then the related timelines have the same hierarchy level and they are said to be hierarchically equivalent.

In this regard, the heuristic we have defined exploits hierarchy information to assign "priority" to the flaws detected during the solving process. The work (Umbrico, Orlandini, and Cialdea Mayer 2015) shows some promising results concerning the application of this heuristic to improve the solving capabilities of EPSL-based planners. In particular, the heuristic can be represented as a *pipeline* of filters that extract the most *promising* flaws to solve first as follows:

$$\Phi^0(\pi) \xrightarrow{f_h} \Phi^1(\pi) \xrightarrow{f_t} \Phi^2(\pi) \xrightarrow{f_d} \Phi^3(\pi) \rightarrow \ \phi^* \in \Phi^3(\pi)$$

Given a partial plan π with an initial set of flaws $\Phi^0(\pi)$, each filter f of the pipeline extract the subset of flaws to solve according to a particular criteria. The first filter f_h applies the hierarchy by selecting the flaws that belong to the most *independent* timelines (i.e. the timeline with the highest hierarchy level). The flaws composing the last set represent equivalent choices from the hierarchy point of view. Thus the planner can randomly selects one of these flaws to solve $\phi^* \in \Phi^3(\pi)$.

Ongoing Works

Currently we are making a comparison of EPSL framework with EUROPA which is one of the most known timeline-based planning framework in the literature. In particular our comparison aims at taking into account different aspects of the planning frameworks and not only their performances. Namely, our goal is to make a deep analysis of thedifferent approaches to timeline-based planning by considering their modeling capabilities, their expressiveness and their solving capabilities, in order to create a shared understanding of the meaning of both timelines and timeline-based plans.

In particular, we are taking into account two real world scenarios by defining the ROVER and the NEPTUS planning domains. Indeed, the selected domains represent two interesting real-world applications that are particularly relevant from the point of view of a plan-based control system. The core of both problems is to model and control a complex system which is able to perform some operations in a specific environment. The plan-based controller must provide the agent with the deliberative capabilities to autonomously synthesize and schedule the sequences of activities needed to perform high-level tasks.

The ROVER planning domain has been extracted from the scenario described on the EUROPA's web site concerning an autonomous exploration rover. This scenario represents a typical and well known application context in AI. It is relevant because it represents a classical single agent control scenario concerning the capability to provide a robotic device with autonomy in order to perform some complex tasks.

Similarly, the NEPTUS planning domain has been extracted from a real-world application senario, described in (Chrpa et al. 2015), where a number of AUVs must gather data about some known underwater phenomena. The problem of controlling an AUV may seem similar to the problem of controlling an autonomous exploration rover. However we have selected this domain for the *coordination* aspect involved. Indeed, in this context, the point is not just to control a single agent, but to safely coordinate several agents (i.e. the AUVs) in order to perform the tasks.

Initial results show relevant difference concerning their modeling approaches w.r.t. the structure of a timeline-based

planning domain, the type of elements the frameworks can model and the way a user must specify constraints to obtain the desired behavior of the system. Moreover there are also relevant differences concerning their interpretation of timeline-based plans and planning solutions.

Moreover I'm currently extending the EPSL planning framework by introducing the capabilities of modeling and reasoning about the temporal uncertainty of a planning domain. Thus, following the proposed formalization the planning framework must be able to model activities whose actual duration cannot be decided by the planner.

References

Barreiro, J.; Boyce, M.; Do, M.; Frank, J.; Iatauro, M.; Kichkaylo, T.; Morris, P.; Ong, J.; Remolina, E.; Smith, T.; and Smith, D. 2012. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *ICKEPS 2012: the 4th Int. Competition on Knowledge Engineering for Planning and Scheduling*.

Borgo, S.; Cesta, A.; Orlandini, A.; Rasconi, R.; Suriano, M.; and Umbrico, A. 2014a. Towards a cooperative knowledge-based control architecture for a reconfigurable manufacturing plant. In *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2014)*. IEEE.

Borgo, S.; Cesta, A.; Orlandini, A.; Rasconi, R.; Suriano, M.; and Umbrico, A. 2014b. A cooperative model-based control agent for a reconfigurable manufacturing plant. In *ICAPS-14, PlanRob - The 2nd ICAPS Workshop on Planning and Robotics*.

Borgo, S.; Cesta, A.; Orlandini, A.; and Umbrico, A. 2015. An ontology-based domain representation for plan-based controllers in a reconfigurable manufacturing system. In *The Twenty-Eighth International Flairs Conference*.

Carpanzano, E.; Cesta, A.; Orlandini, A.; Rasconi, R.; Suriano, M.; Umbrico, A.; and Valente, A. 2015. Design and implementation of a distributed part-routing algorithm for reconfigurable transportation systems. *International Journal of Computer Integrated Manufacturing* 1–18.

Cesta, A., and Fratini, S. 2008. The Timeline Representation Framework as a Planning and Scheduling Software Development Environment. In *PlanSIG-08. Proc. of the 27th Workshop of the UK Planning and Scheduling Special Interest Group, Edinburgh, UK, December 11-12*.

Cesta, A.; Cortellessa, G.; Fratini, S.; and Oddi, A. 2009. Developing an End-to-End Planning Application from a Timeline Representation Framework. In *IAAI-09. Proc. of the 21st Innovative Application of Artificial Intelligence Conference, Pasadena, CA, USA*.

Chien, S.; Tran, D.; Rabideau, G.; Schaffer, S.; Mandl, D.; and Frye, S. 2010. Timeline-Based Space Operations Scheduling with External Constraints. In *ICAPS-10. Proc. of the 20th Int. Conf. on Automated Planning and Scheduling*.

Chrpa, L.; Pinto, J.; Ribeiro, M. A.; Py, F.; Sousa, J.; and Rajan, K. 2015. On mixed-initiative planning and control for autonomous underwater vehicles. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, 1685–1690.

Cialdea Mayer, M.; Orlandini, A.; and Umbrico, A. 2014. A formal account of planning with flexible timelines. In *The 21st International Symposium on Temporal Representation and Reasoning (TIME)*, 37–46. IEEE.

Cialdea Mayer, M.; Orlandini, A.; and Umbrico, A. 2015. Planning and execution with flexible timelines: a formal account. *Acta Informatica* 1–32.

Laborie, P., and Ghallab, M. 1995. Ixtet: an integrated approach for plan generation and scheduling. In *Emerging Technologies and Factory Automation, 1995. ETFA '95, Proceedings., 1995 INRIA/IEEE Symposium on*, volume 1, 485–495 vol.1.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed., *Intelligent Scheduling*. Morgan Kauffmann.

Stefano, B.; Amedeo, C.; Andrea, O.; and Alessandro, U. to appear. A planning-based architecture for a reconfigurable manufacturing system. In *The 26th International Conference on Automated Planning and Scheduling*.

Umbrico, A.; Orlandini, A.; and Cialdea Mayer, M. 2015. Enriching a temporal planner with resources and a hierarchy-based heuristic. In *AI*IA 2015, Advances in Artificial Intelligence*. Springer International Publishing. 410–423.

Dissertation Abstract

Emre Ökkeş Savaş
Supervisors: Maria Fox, Derek Long
Department of Informatics,
King's College London, London, WC2R 2LS, UK
e-mail: okkes.savas@kcl.ac.uk

Abstract

This dissertation outlines the work I have done since the beginning of my research degree. My research interest is in constrained resource planning, where I am particularly interested in the applications of operations research techniques in the task planning. Planning community has started to use operations research tools in their work in recent years. I aim to introduce new techniques to the task planning, which are widely practiced in operations research. The contribution of this paper is to present a generalisation of variables in the planning domain. We consider all types of predefined variables and the duration to a new type, which we call *control parameters*. We also describe the development of our new planner POPCORN (Partial-Order Planning with Constrained Real Numerics) that can reason with control parameters. We present an example of how existing task planning benchmark domains can be extended to develop enriched plans. We also provide an example to demonstrate the robustness and applicability of our approach.

1 Introduction

Integration of the temporal and metric fluents has become a popular field of research in the task planning. Many off-the-shelf planners (Della Penna et al. 2009; Fernández-González, Karpas, and Williams 2015a; Bajada, Fox, and Long 2015; Bryce et al. 2015; Piacentini et al. 2015) have overcome challenges posed in hybrid systems with the help of some optimisation tools. Finding the timestamps and the durations of actions have been the major interest of such planners, while the remainder dynamics of the real-world problems are neglected. The duration of an action is the only variable in PDDL domains, for which the planner has the freedom to assign a value. However, there are numerous time-independent dynamics in real-world problems. For instance, the driver decides on the initial velocity of the vehicle, the refuel amount before or during the journey. These dynamics are assigned with a fixed value at the initial state in PDDL problem instances, but the planner should not be constrained with such discretised values. In this paper, we present an approach to include variables other than the duration of the action in planning domains. We consider generalising all sorts of variables into a new type, which we call *control parameters*. We consider the duration of an action as a special type of control parameter, where it plays an important role in the plan ordering.

```
(:durative-action refuel
  :parameters (?v - vehicle ?l - location)
  :control (?fuel - number)
  :duration (= ?duration 10)
  :condition (and
    (at start (>= ?fuel 0))
    (at start (<= ?fuel (fuel-max ?v)))
    (at start (at ?v ?l))
    (over all (at ?v ?l))
    (at start (has-petrol-station ?l)))
  :effect (at end
    (increase (fuel-left ?v) ?fuel)))
```

Figure 1: Updated refuel action in Transport-numeric domain

Many state-of-the-art planners only find a sequence of the time-stamped actions to reach a goal state. Additionally, the use of control parameters enables a planner to make a decision about the values of the variables predefined in the planning domain. The planner can constrain the feasible region of a control parameter during the plan construction. We can present a simple example here, based on the *transport-numeric* domain, which is used as a benchmark domain in the International Planning Competition in 2008[1]. In this domain, trucks deliver packages from their initial locations to the goal locations. The fuel level of trucks decreases as trucks move from one location to another. The `refuel` durative action fills the tanks of trucks to the full tank (a fixed value), even if the truck only needs a small amount of petrol to reach its goal location. It would be more realistic if the planner does not assign a fixed numeric value to the fuel level, but assigns a value that is sufficient to reach its goal. This value can be constrained by numeric constraints that are *dynamically developed* during planning. Figure 1 shows the updated version of the `refuel` action where the refuel amount is taken as a control parameter: `?fuel`, where it is constrained within the real numeric region of `[0, (fuel-max ?v)]` (`(fuel-max ?v)` state variable is fixed with a constant value at the initial state).

In this paper we present our new planning system, POPCORN, that can reason with control parameters with the

[1]Original domain used in the competition can be obtained from *http://ipc08.icaps-conference.org/deterministic/domains.html*

help of linear programming (LP). The implementation we have so far focussed on linear constraints, however we will extend this concept to a non-linear case in the future. POPCORN is built on POPF (Coles et al. 2010), so that the existing sophistication of a temporal planner is preserved. The main objective of this paper is to describe the major steps taken to develop our new planning system. The structure of this paper is in the following order. We consider the related work in the field in Section 2. We then provide a simple example in Section 3 to use throughout the paper. We carry on with the description of POPF planner, which constitutes the basis of our implementation, in Section 2. We then describe the required modifications to the existing problem formulation, constructing linear program (LP), the heuristic guidance, and the forward state space search. Finally, we provide the future work and the preliminary evaluation of our approach.

2 Related Work

Early work exploring the use of the control parameters in planning domain is considerably limited. Kongming planner (Li and Williams 2008) captures the interaction of the dynamic continuous variables with *flow tubes* produced at each action layer. The flow tubes contain control trajectories of the variables as the graph expands over time. It can only handle problems with linear effects. In order to capture the continuous dynamics of a problem, time is discretised while the rate of change is taken as a variable. This concept contrasts with COLIN (Coles et al. 2009) and POPF planners, where the duration is taken as a variable, while the rate of change remains constant. Kongming suffers from the limitation of the number of happenings in the plan, so it fails generating plans requiring long time horizons.

Enrique Fernández-González, Erez Karpas and Brian Williams have recently studied the planning with continuous control parameters (2015a). Their work has considered the main stages in the development of the Scotty planning system. The Scotty planner combines the flow tube representation of the Kongming with the forward-chaining search and the linear programming used in the COLIN planner. The flow tubes are used to capture continuous effects with control parameters. It uses the forward search to overcome the happening limitation of Kongming. The planner finds *a fixed plan*, in which the planner assigns a value to the control parameters at an early stage during planning, which makes the plan invalid due to early-commitment. Therefore, Scotty finds *a flexible plan*, in which it leaves the decision of the values of control parameters to an executive during plan execution in order not to invalidate the plan. On the other hand, the planner assigns values to the timestamps of actions without any interaction with an executive. In addition, Scotty does not support discrete numeric change (Fernández-González, Karpas, and Williams 2015b). Our planner, however, makes a decision for the values of control parameters without any human interaction, and it *delays* the valuation of these parameters, including `?duration` and the timestamps of actions, until a decision is forced by the planner.

3 A Motivating Example

We now present a simple example to introduce the control parameters. Suppose that we are planning to go to a pub. Initially, we are at home and have only £2 in our pocket. We aim to be at the pub with £20 in our pocket and to have already bought snacks on the way to the pub. Intuitively we would withdraw sufficient cash to buy snacks and to have £20 at the pub. We would not want to withdraw more or less cash than required when at the cash point. There are three ATM machines at the cash point. Each machine has a limited balance available that can be withdrawn (`(balance ?m)`), and minimum withdrawal amount is £3. Actions of this domain and the initial/goal states are given in Figure 3 and 2, respectively. In addition, Figure 2 shows the metric objective of the problem that plays an important role in the valuation of the control parameter `?cash` . The language we use is a modified version of PDDL 2.1 to encode control parameters. We list all control parameters except `?duration` in a new line, `:control()`, in a durative action.

```
 (:init (at person1 home)
  (canbuy person1 store)
  (canwithdraw person1 cashpoint)
  (available)
  (located atm1 cashpoint)
  (located atm2 cashpoint)
  (= (inpocket person1)  2)
  (= (balance atm1) 50)
  (= (balance atm2) 100)
  (= (balance atm3) 150))

 (:goal (and (>= (inpocket person1) 20)
(gotsnacks person1)  (at person1 pub))
 (:metric minimize (inpocket person1)))
```

Figure 2: The initial state of the cash point problem.

In this example, the amount of cash we want to withdraw, `?cash`, depends on which actions we apply after visiting the cash point. Early assignment of the value of a control parameter may lead to generate poor plans. For instance, assigning a value to `?cash` before buying snacks would result in visiting the cash point twice. Therefore, the decision of withdrawal amount should be made at a later stage in the plan (or eventually, at the end of the plan). POPCORN builds up all the linear constraints acting upon the control parameter `?cash` until the end of the plan. Then, the planner calls the linear program to optimize all variables, i.e. `?cash`, subject to the metric objective of the problem. Since the metric objective is to minimize `inpocket` state variable in this example, the planner chooses the minimum bound of this variable as its value.

4 Background

Temporal and numeric planning has been emerged together with the help of linear programming. Numeric and temporal constraints are handled separately in the early instances of temporal planners (Coles et al. 2008). Integration of the

```
(:durative-action WithdrawCash
:parameters (?p - person ?a - location
?m - machine)
:control (?cash - number)
:duration (= ?duration 2)
:condition (and (over all (at ?p ?a))
   (at start (>= ?cash  3))
   (at start (<= ?cash (balance ?m)))
   (at start (canwithdraw ?p ?a))
   (at start (located ?m ?a) ) )
:effect (and
  (at start (decrease (balance ?m) ?cash))
  (at end (increase (inpocket ?p) ?cash))))

(:durative-action BuySnacks
:parameters (?p - person ?a - location)
:duration (= ?duration 1)
:condition (and   (at start (at ?p ?a))
  (over all (at ?p ?a))
  (at start (>= (inpocket ?p) 5))
  (at start (canbuy ?p ?a)))
:effect (and (at start (not (available)))
  (at end (decrease (inpocket ?p) 3))
  (at end (gotsnacks ?p))))
```

Figure 3: Main actions of the cash point domain.

temporal and numeric constraints together made it possible to handle continuous numeric change, in which the value of a state variable can depend on the timestamp and the duration of the action (Coles et al. 2012). In order to implement temporal-numeric planning with control parameters we built our planning system on the POPF planner, which can already handle this integration. In general, the state representation of a temporal-numeric planning problem can be shown by a tuple S = $\langle F, V, Q, P, C\rangle$, where:

- F is the set of propositions that are true in the current state S.
- V is the vector of values of the numeric state variables. Depending on the length of a state S, the state variable V varies within V^{min} and V^{max} due to linear continuous numeric effects.
- Q is a list of actions, which are started but not yet finished.
- P is the plan to reach the current state S.
- C is a list of temporal constraints accumulated over the steps in P.

In addition to the state representation given above, POPF includes further elements[2] to support partial-order planning. The partial-order mechanism simply minimises the ordering constraints to avoid early-commitment during forward search in order to achieve flexible plans. The temporal constraints are added as they are needed to meet the preconditions of actions in a possible plan.The existing partial-order mechanism of POPF helps POPCORN to avoid early-commitment of assigning values to the control parameters. As discussed in Section 3, the early-commitment in the valuation may lead to generate poor plans.

[2]Full list of partial ordering extensions to state representation for propositional and numeric case can be found in (Coles et al. 2010)

LP Temporal and Numeric Scheduling

The POPF planner inherits the use of linear programming from the COLIN planner. It uses the LP to check the temporal and the numeric consistency of a state. The state variables that capture discrete/continuous numeric change along the trajectory of the plan are defined as follows:
Each $v_i \in V_i$ records the value of each state variable v *just before* the step i. Similarly, each $v'_i \in V'_i$ records the value of a state variable v *immediately after* the step i. For instance, a state variable v can have a discrete instantaneous numeric change at a step i. In this case, $v'_i = v_i + c$ constraint, where v_i is increased by the numeric value of c at the step i, is added to the LP in order to record this change. Table 1 shows the constraints and variables created to record numeric changes over the control parameter. bal_i and inp_i represents the $(balance\ ?m)$ and $(inpocket\ ?p)$ state variables at step i, respectively.

Plan Action	LP Variable	[lb, ub]	Constraints
Withdraw (start)	$cash$	[3, inf]	$cash \geq 3$
		[0,50]	$cash \leq bal_0$
	bal_0	[50,50]	bal_0
	bal'_0	[0,47]	$= bal_0 - cash$
Withdraw (end)	$cash$	[3,50]	$cash$
	inp_1	[2,2]	inp_1
	inp'_1	[5,52]	$inp_1 + cash$
BuySnacks (start)	inp_2		$= inp'_1$
	inp'_2	[5,52]	≥ 5
BuySnacks (end)	inp_3	[5,52]	$= inp'_2$
	inp'_3	[2,49]	$= inp_3 - 3$

Table 1: Variables and constraints acting upon `?cash` parameter, that are collected from the initial state to reach the goal state. [*lb, ub*] represents the upper and lower bound limits of the variables at a state.

The use of LP makes it possible to record numeric change *between* steps. This change can be considered as a continuous change, because the time elapsed between steps is a variable. In this case, $v'_{i+1} = v'_i + \delta v_i(step_{i+1} - step_i)$ is added to the LP that records the continuous linear numeric change between consecutive steps. $step_i$ is the timestamp LP variable of step i, while δv_i represents the gradient of continuous change on v at step i. The value of a state variable v depends on time elapsed before the next action is executed. We can then say the value of state variable v varies between lower ($v^{min} \in V^{min}$) and upper ($v^{max} \in V^{max}$) bounds, which are required to check state validity when the action is still executing. In order to compute these bounds, additional variables, v_{now} and $step_{now}$, are added to the LP to check the state validity. Figure 4 shows the relationship discussed between temporal, numeric variables encoded into the LP.

5 Planning with Control Parameters

The main distinction between POPF and POPCORN planners is that POPCORN can reason with variables other than the duration in planning. We consider all variables available in the planning domain as control parameters. This is

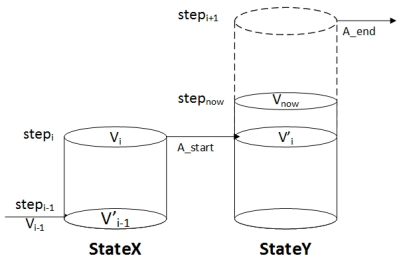


Figure 4: Schematic representation of the relationship between numeric state variable, V, and timestamp variable that are encoded in LP. V_{now} is used to compute upper-lower bounds of V during continuous linear numeric change.

achieved by extending the existing machineries of the POPF. We consider the details of each component in their related subsections. In summary, we extend the existing problem definition to capture the control parameters defined in actions. We define the additional constraints and variables added to the LP based on the numeric preconditions and effects. We provide the modifications made to the existing heuristic approach of POPF and analyse the effects of the control parameters to the search space.We use the cash point example to enumerate elements discussed in the related subsections.

Problem Definition

Many state-of-the-art temporal planners make a decision only about *which* actions to apply, and *when* to apply these actions. Our new planner can additionally make a decision about the values of the predefined numeric variables, which are constrained with linear constraints in the linear program. Slightly different than the existing state representation of POPF, the new state representation for temporal-numeric planning with control parameters problem can be shown by a tuple S = $\langle F, V, Q, P, D, L\rangle$, where:

- F is the set of propositions that are true in the current state S.
- V is the vector of values of the numeric state variables. Depending on the length of a state S, the state variable V varies within V^{min} and V^{max} due to continuous or control parameter numeric effects.
- Q is a list of actions, in which actions started but not yet finished.
- P is the plan to reach the current state S.
- D is a list of all control parameters available, including the `?duration` variable in durative actions, in the planning domain, where each $d \in D$ is a tuple $\langle op, i, num\rangle$:
  - op is the identifier of instantiated action,
  - i is the index of the step in the plan,
  - num is the unique identifier of each $d \in D$,

  where the corresponding control parameter(s) are added. The unique identifier, num, of duration variable is identical in every durative action, since there is only one duration variable defined in an action.
- L is a list of constraints that encapsulates discrete numeric change with the control parameters over the steps in P, where each $l \in L$ of the form $minControl(d) \leq d \leq maxControl(d)$. The value of d lies within a range of values constrained by $minControl(d)$ and $maxControl(d)$. These bounds on d are determined from the numeric preconditions of the action on d. If there is not any numeric precondition in d defined, the range of d is set to $[0, inf]$. We restrict our definition of d to be a positive real number in order to avoid modeling errors due to sign convention in the domain.

Checking Plan Consistency with LP

In this section we consider additional variables and constraints added to the LP to support control parameters. Before we begin the formulation, it is worthwhile mentioning the main characteristic of control parameters within an action instance. The control parameter is a *local* variable, whose scope is limited to the action it is defined. It can be carried out through the plan with the numeric state variables. The following equation gives the relationship between these state variables v, and control parameters.

In general form, where step k is the current state:

$$v_{i+1} = v_{val} + \sum_{n=0}^{num} \delta w_{i,n} d_{i,n} \quad (1)$$

where,

- $d_{i,n}$ is the n^{th} control parameter defined in the action op applied to the plan at $step_i$.
- v_{val} is the variable that contains the most recent numeric value of v prior to v is affected by the control parameter d. If there is no discrete numeric effect on v at $step_i$, the value of v_{val} is equal to v_i
- $\delta w_{i,n}$ is the total gradient of the n^{th} control parameter acting upon v.
- v_{i+1} is the value of the numeric state variable immediately after the discrete control parameter effect acting on v

Temporal and numeric constraints are added to the LP to confirm that the plan to reach a state S can be scheduled. In our new planner POPCORN, the constraints with control parameters are used to check whether there is a $feasible$ range of values of the control parameter that can satisfy the plan to reach state S. In order to capture these constraints, the following constraints below are added to the LP.

- Any numeric precondition that is given in the form:
  $\langle v, sgn, \mathbf{w}\cdot\mathbf{v}+k.(d_{i,n}) + c\rangle$, s.t. $sgn \in \{\leq, <, =, \geq, >\}$
  $\langle d_{i,n}, sgn, v\rangle$, s.t. $sgn \in \{\leq, <, \geq, >\}$
  $\langle d_{i,n}, sgn, c\rangle$, s.t. $sgn \in \{\leq, <, \geq, >\}$

- Any numeric effect that is in the form: $\langle v, sgn, \mathbf{w}\cdot\mathbf{v}+k.(d) + c\rangle$, s.t. $sgn \in \{+=, -=, =\}$; $c, k \in \mathbb{R}$

are added as constraints over V to the LP. If the constructed LP with these constraints is not solvable, then the state S is pruned from the search space, and the planner backtracks to look for a state, in which the LP has a feasible solution. Our approach isolates the nonlinear interaction between a control parameter and a temporal variable. The LP is inadequate to

check state consistency for nonlinear states. We are working on addressing this with the help of an appropriate nonlinear solver.

Temporal-Numeric State-Space Search

Duration of an action in a durative action may not be fixed, and it can be determined by either the values of metric fluents: i.e. `(<= ?duration (v ?p))`, or it is constrained within a range of values (Coles et al. 2009), i.e. `(and (>= ?duration 10) (< ?duration 50))`. Likewise, the value of control parameter defined in an action is not fixed, but it can be constrained within some interval. Similar logic applies to the numeric state variables that have ever had a discrete control parameter dependent change. The value of a state variable is constrained within a range of values, $[v^{min}, v^{max}]$, that the planner has the freedom to choose. Figure 5 shows the differences between discrete, continuous, and discrete control parameter changes acting upon state variable v. Suppose that state variable v is affected by discrete numeric changes at $step_1$ and $step_4$, linear continuous change between $step_2$ and $step_3$, and discrete change with control parameter at $step_3$. v can take at any numeric value within dashed area.

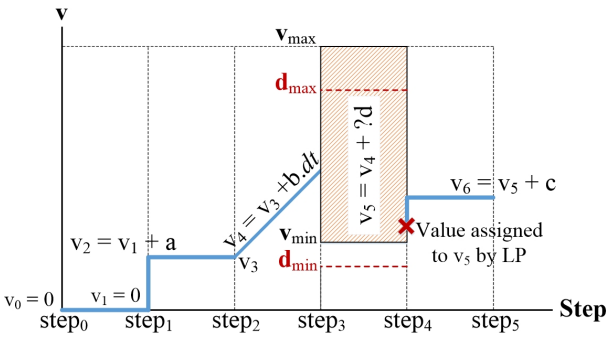


Figure 5: Schematic representation of numeric state variable, v, affected by discrete, continuous, discrete control parameter numeric change over steps 0 to 5. Shaded area shows the feasible region of values of v that can be assigned by calling linear program. $?d$ is a control parameter with a value of d lies within range of $[d_{min}, d_{max}]$. $a, b, c \in \mathbb{R}$.

The existence of control parameters generate a complex branching choice in the search space, while there remains a finite set of action choices available in the search space. Then we can say the width of the search tree remains the same, while the depth of the search tree dramatically increases after a control parameter effect. Figure 6 illustrates this effect in the search space for our cash point example. (*inpocket* += *?cash*) effect produces infinitely many states, because the value of *?cash* is not yet assigned. However, if our implementation is forced to branch over this infinite space, it avoids this by leaving the choice to the LP constraint space. avoids this by leaving the decision to the LP constraint space.

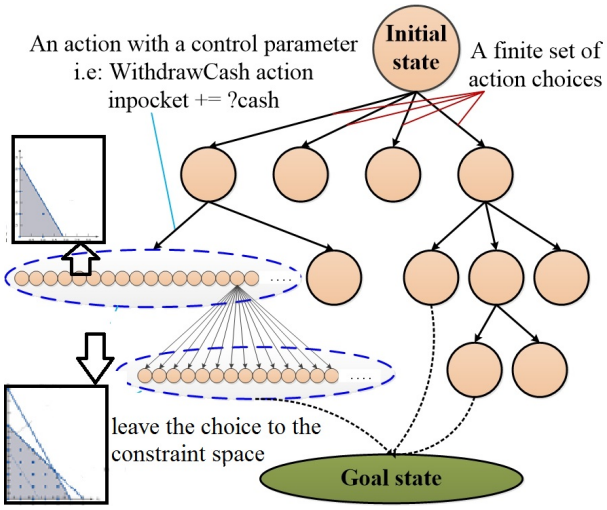


Figure 6: Schematic representation of the search space where there is a control parameter effect. The nodes represent the state reached, and the edges represent the action applied to reach the next state. The graphs in black boxes represent the LP constraint space, which is used to avoid complex branching choice.

Modifications to The Temporal RPG Heuristic

The Metric Relaxed Planning Graph (Hoffmann 2003) heuristic has been widely used in the numeric planning over the last decade. POPF planner uses a heuristic, Temporal RPG, to guide the planner in the search space towards the goal. The Temporal RPG (TRPG) heuristic is a modified version of the Metric RPG. The main difference between two heuristics is the timestamps associated to each action and fact layer in TRPG.

Our modification to the existing temporal RPG heuristic of POPF is to make an optimistic assumption: If an action a has a control parameter effect on a variable v, then the control parameter is relaxed to whichever $minControl(d)$ or $maxControl(d)$ gives the largest(smallest) effect. In case the $minControl(d)$ and/or $maxControl(d)$ depend on a the value of a state variable (i.e: `(<= ?cash (balance ?m))`), then the heuristic calls the LP, which only contains the time-independent numeric constraints of the action, to precompute the bounds for the heuristic before extracting a relaxed plan. For instance, in the reachability analysis, the following LP constructed to find the upper bound of *?cash*:

Maximise: $?cash$
Subject to:
$bal_0 = 50$
$?cash \geq 3$
$?cash - bal_0 \geq -inf$
$?cash - bal_0 \leq 0$
$?cash + bal'_0 - bal_0 = 0$
$inp_0 = 2$
$inp'_0 - inp_0 - ?cash = 0$

6 Evaluation

In this section we present the preliminary results of our implementation. Since the existing planners that reason about control parameters are not available online, we compare the capability of our implementation with our base planner. However, POPF can not run at all on the cash point problem if we do not provide a fixed withdrawal value. Therefore, we fix the withdrawal value, `?cash`, at £10 for running the experiments with POPF. Regardless of the value we fix the withdrawal value to, the POPF will always generate longer plans. The POPCORN does not require any fixed value, so it is able to solve the problem for any value of `(inpocket ?p)` within the bounds defined.

We compared the performance of POPF and POPCORN in problems, where the goal `(>= (inpocket person1) 500)` incrementally increases to `(>= (inpocket person1) 950)` in every problem instance. We observe lengthy plans produced by POPF due to repetitive `WithdrawCash` actions. Table 2 shows the results of this evaluation. This preliminary evaluation shows that our approach dramatically decreases the number of states evaluated and the plan length produced by our based planner. POPF is not able to produce plans for `(>= (inpocket person1) 900)` problem instances, because it runs out of memory.

	# States Evaluated		Plan Length	
inpocket$\geq$	POPF	POPCORN	POPF	POPCORN
500	508	457	57	10
550	2089	142	62	10
600	2248	484	67	12
650	3892	484	72	12
700	7461	484	77	12
750	14143	1430	82	13
800	8750	1430	87	13
850	32341	1430	92	13
900	–	4366	–	14
950	–	4366	–	14

Table 2: Number of states evaluated and plan length evaluation of POPF and POPCORN planners, where `inpocket` goal is discretised for POPF.

7 Future Work

I consider finalising the implementation of POPCORN presented in this paper, and extend its capability by implementing a nonlinear solver to solve problems requiring nonlinear numeric change. In order to achieve this, I will initially identify which nonlinear solver is sufficient to use in planning. Then, I will explore the required modifications to implement this solver within our existing planning system. Another future work I want to work on is about managing the preferences of objectives defined in the domain. I plan to use Goal-Programming approach to minimise the penalty costs for the multi-objective planning domains. The minimised penalty cost can be used to get guidence in the search space as a tie-breaking factor (where the timespan of plan options are equal). Finally, I consider extending the generalisation of the control parameters with non-numeric object variables. As mentioned in Section 1, the planner is initially constrained with discretised assignments, for which the planner actually should have freedom to choose. This approach can be implemented for objects defined in planning problem.

8 Conclusion

Physical and logical properties of the real-world examples require multiple numeric variables to create realistic planning models. In this paper we provide the preliminary work of our implementation to handle control parameters in the planning domain. We generalise the use of all parameters to a new type to fully integrate temporal and numeric planning. At this stage we identified the necessary modifications to the existing mechanism of our base planning system.

References

Bajada, J.; Fox, M.; and Long, D. 2015. Temporal planning with semantic attachment of non-linear monotonic continuous behaviours. In *Proceedings of the 24th International Conference on Artificial Intelligence*, 1523–1529. AAAI Press.

Bryce, D.; Gao, S.; Musliner, D.; and Goldman, R. 2015. Smt-based nonlinear pddl+ planning. In *Proceedings of the Twenty Nineth Conference on Artificial Intelligence (AAAI-15). AAAI Press*.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. Planning with problems requiring temporal coordination. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 08)*.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2009. Temporal planning in domains with linear processes. In *Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 42–49.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2012. Colin: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research* 1–96.

Della Penna, G.; Magazzeni, D.; Mercorio, F.; and Intrigila, B. 2009. Upmurphi: a tool for universal planning on pddl+ problems. In *Nineteenth International Conference on Automated Planning and Scheduling*.

Fernández-González, E.; Karpas, E.; and Williams, B. C. 2015a. Mixed discrete-continuous heuristic generative planning based on flow tubes. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Fernández-González, E.; Karpas, E.; and Williams, B. C. 2015b. Mixed discrete-continuous heuristic generative planning based on flow tubes. In *Proceedings of the 3rd Workshop on Planning and Robotics (PlanRob-15)*, 106–115.

Hoffmann, J. 2003. The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables. *Journal of Artificial Intelligence Research* 291–341.

Li, H. X., and Williams, B. C. 2008. Generative planning for hybrid systems based on flow tubes. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 206–213.

Piacentini, C.; Alimisis, V.; Fox, M.; and Long, D. 2015. An extension of metric temporal planning with application to ac voltage control. *Artificial Intelligence* 229:210–245.

Planning with PDDL3.0 Preferences by Compilation into STRIPS with Action Costs

Percassi Francesco
University of Brescia
Department of Information Engineering
f.percassi@unibs.it

Abstract

The research community has sought to extend the classical planning problem following two strategies. The first one follows a top-down approach consisting in the development of solvers that support a more general class of problems; the second one follows a bottom-up approach consists in extending the applicability range of current classical planners. A possible interesting approach consists in compiling the new features offered by recent extension of planning language into a simpler target language such as STRIPS or ADL. PDDL 3.0, the official language in 2006 fifth IPC, introduced state-trajectory constraints and preferences in order to better characterize the solution quality. In this work I present a compilation schema, inspired by some previous works, for traslating a STRIPS problem enriched with all kind of PDDL 3.0 preferences into an equivalent STRIPS problem with action cost.

Introduction and background

Given a problem described by an action domain, an intial state and a description of goal state, the aim of classic planning paradigm is finding a sequence of actions that can transform, if they are performed, the initial state into the target state. It is possible to distinguish which solution is preferable among the set of possible solutions evaluating the plan cost as the number of its actions. This approach has been extended to the minimal plan cost evaluated as the sum of the cost assigned to each contained action. A more sophisticated recent approach for characterizing when a solution is better than others is based on the notion of preference, which are properties that a plan has to satisfy to increase its quality. Planning with preferences concerns the generation of plans for problems involving soft goal or soft state-trajectory constraints, called preference in PDDL 3.0 (Gerevini et al. 2009), which are preferable to satisfy, but that they are not necessary to hold in a valid plan.

In PDDL 3.0 has been proposed some new features in order to increase the expressive power about the quality solution specification. The new introduced constructs include *soft state-trajectory constraints*, which are constraints that should be satisfied in the state trajectory to increase the quality plan, and *soft problem goal*. An approach to assign a priority to each *preference* (hereafter we indicate indistinctly a soft goal or a soft state-trajectory as preference) consists into penalizing their violation with a real value that is used to decrease the plan metric.
In PDDL 3.0 the following class of preferences can be expressed:

- *always*, which requires that a condition should hold in every reached state; this kind of preferences is very useful to express safety or maintenance conditions;
- *sometime-before*, which requires that a condition Ψ has become true before a second condition Φ becomes true;
- *sometime*, which requires that a condition becomes true at least once in the state trajectory of the plan;
- *at-most-once*, which requires that a condition becomes true at-most-once once in the state trajectory of the plan;
- *soft goal*.

This work describes a compilation scheme which is an extension of what proposed in (Ceriani and Gerevini 2015) where only always preference and soft goal are considered.

STRIPS+ with preferences

A STRIPS+ problem is a tuple $\langle F, I, O, G, c\rangle$ where $\langle F, I, O, G\rangle$ is a STRIPS problem and c is a function that maps each $o \in O$ to a non-negative real number. The cost of a plan π is defined as $c(\pi) = \sum_{i=0}^{|\pi|-1} c(a_i)$, where $c(a_i)$ represents the cost of the i-th action a_i in π and $|\pi|$ is the plan lenght. Without loss of generality, we will assume that the condition of a preference P_i is expressed in conjuctive normal form, for example $P_i = p_1 \wedge p_2 \wedge ... \wedge p_n$, where each p_j with $j \in [1, ..., n]$ is a clause of P_i formed by literals over the problem fluents. We write $\pi \models_{typ} P_i$ to indicate that plan π satisfies a typ preference P_i where typ indicate its type among $\{a, sb, st, amo, sg\}$ which abbreviating always, sometime-before, sometime, at-most-once and soft goal.

Definition 1 *A STRIPS+ problem with preferences is a tuple $\langle F, I, O, G, \mathcal{P}, c, u\rangle$ where:*

- *$\langle F, I, O, G, c\rangle$ is a STRIPS+ problem;*

- $\mathcal{P} = \{AP \cup SBP \cup STP \cup AMOP \cup SG\}$ *is the set of the preferences of* Π *where* AP, SBP, STP, $AMOP$ *and* SG *contain respectively always, sometime-before, sometime, at-most-once and soft goal prefences;*
- u *is an utility function mapping each* $P \in \mathcal{P}$ *to a value in* $\mathbb{R}_0^+$

In the following the class of STRIPS+ with a set of preferences is indicated with STRIPS+P.

Definition 2 *Let* Π *be a STRIPS+P problem with a set of different kind of preference* $\mathcal{P}$*. The utility* $u(\pi)$ *of a plan* π *solving* Π *is the difference between the total amount of utility of the preferences by the plan and its cost* $u(\pi) = \sum_{P \in \mathcal{P}:\pi \models_{typ(P)} P} u(P) - c(\pi)$ *where typ is a function that map each* $P \in \mathcal{P}$ *to the respective type, i.e.* $typ : \mathcal{P} \rightarrow \{a, sb, st, amo, sg\}$

A plan π with utility $u(\pi)$ for a STRIPS+P problem is optimal when there is no plan π' such that $u(\pi') > u(\pi)$. The definitions below are introduced to simplify the notation in the discussion.

Definition 3 *Given a preference clause* $p = l_1 \vee l_2 \vee ... l_n$*, the set* $L(p) = \{l_1, l_2, ..., l_n\}$ *is the equivalent set-based definition of* p *and* $\overline{L}(p) = \{\neg l_1, \neg l_2, ..., \neg l_n\}$ *is the literal complement set of* $L(p)$*.*

Definition 4 *Given an operator* $o \in O$ *of a STRIPS+P problem,* $Z(o)$ *is the set of literal defined as:* $Z(o) = (prec(o) \setminus \{p \mid \neg p \in eff(o)^-\}) \cup eff(o)^+ \cup eff(o)^-$*. Note that set* $Z(o)$ *represents the literals certainly true in the state resulting from the application of operator* o*.*

Preferences and Class of Operators

In our compilation scheme of a STRIPS+P problem we have to distinguish, for each kind of preference, different class of operators that are specified in the following definitions. This distinction is important in order to specialize the operators compilation based on how they interact with the preferences of the problem.

Definition 5 *Given an operator* o *and CNF formula* Φ *of a preference* P *of a STRIPS+P problem, we say that* o *can make true* Φ *if:*

- *there is at least a clause* φ *of* Φ *such that* $L(\varphi) \cap Z(o) \neq \emptyset$ *and* $L(\varphi) \not\subseteq prec(o)$*; we indicate the set of clause which satisfy this condition as* $C(o, \Phi)$*; the complementary set of the remaining clauses is defined as* $\overline{C}(o, \Phi) = \{\varphi \in \Phi \mid \varphi \notin C(o, \Phi)\}$
- *for each clause* $\varphi \notin C(o, \Phi) \Rightarrow \overline{L}(\varphi) \not\subseteq Z(o)$*.*

The first condition in Definition 5 requires that exists at least a clause of the formula which contains some literals that become certainly true in the state resulting from the execution of o and that this clause is not true in the state where o is applied. The second condition requires that the other clauses of the formula, which are not contained in $C(o, \Phi)$, are not falsified in the resulting state from the application of o.

Always

An always preference has the following PDDL syntax $(always\ \Phi)$ where the formula Φ has to hold in each reached state of the plan.

Definition 6 *Given an operator* o *and an always preference* $P = (always\ \Phi)$ *of a STRIPS+P problem,* o *is a* ***violation*** *of* P *if there is a clause* ϕ *of* Φ *such that:* $\overline{L}(p) \subseteq Z(o) \wedge \overline{L}(p) \not\subseteq prec(o)$*.*

If an operator violates a preference, the preference is unsatisfied independently from the state resulting from the application of the operator.

Definition 7 *Given an operator* o *and a always preference* P *of a STRIPS+P problem,* o *is a* ***threat*** *of* P *if it is not a violation and there exists a clause* p *of* P *such that:* $\overline{L}(p) \cap Z(o) \neq \emptyset \wedge L(p) \cap Z(o) = \emptyset \wedge \overline{L}(p) \not\subseteq prec(o)$

A clause p of P satisfying the condition of the definition above is a *threatened clause* of P. A threatened preference (clause) may be falsified by an operator depending on the state where the operator is applied. The expression $\overline{L}(p) \not\subseteq prec(o)$ in Defintion 5-6-7 is necessary to avoids that an operator o is considered a violation/threat when its precondition is already violated in the state where it is applied. The set of always preferences of Π which are threatened/violated by the operator o are denoted respectively $T_{ag}(o)$ and $V_{ag}(o)$.

Definition 8 *Given an operator* o *and a always preference* P *of a STRIPS+P problem,* o *is a* ***safe*** *for* P *if:*

- *for all clauses* p *of* P*,* $L(p) \cap Z(o) \neq \emptyset$ *or* $\overline{L}(p) \cap Z(o) = \emptyset$ *holds;*
- *there exists a clause* p *such that* $\overline{L}(p) \subseteq prec(o)$*.*

Sometime-Before

A sometime-before constraint has the following PDDL syntax $(sometime\text{-}before\ \Phi\ \Psi)$, which in the following we abbreviate with $\langle \Phi, \Psi \rangle$. The meaning of $\langle \Phi, \Psi \rangle$ is that if Φ is true in a state s then Ψ must have been true in state before s.

Definition 9 *Given an operator* o *and a sometime-before preference* $P = \langle \Phi, \Psi \rangle$ *of a STRIPS+P problem,* o *is a* ***potential support*** *for* P *if* o *can make* Ψ *true.*

An operator that satisfied Definition 9 is a *potential* support because its behaviour respect to the interested preference depends by the state where o is applied and consequently from the resulting state. We can distinguish two situations:

- if formula Ψ of P does not become true in the resulting state, then o is a *neutral* operator;
- if P is not violated in the state s where o is applied and the formula Ψ of P becomes true in the resulting state, then o is a *real support* operator.

The compilation scheme must take account of both these possibilities.

Definition 10 *Given an operator* o *and a sometime-before preference* $P = \langle \Phi, \Psi \rangle$ *of a STRIPS+P problem,* o *is a* ***potential threat*** *for* P *if* o *could make true* Φ*.*

Similarly to definition 9 also in this case the potential threat defines its behavior in correspondence of the consequences of its application. We distinguish the following situations:

- if formula Ψ of P does not become true in the resulting state, then o is a *neutral* operator;
- if formula Ψ of P becomes true in the resulting state and the formula Φ has become true at least once in a earlier state, than s is *neutral* otherwise if the formula Φ has never become true, then o is a *violation*.

The set of sometime-before preferences of Π which are potentially threatned/supported by the operator o are denoted respectively with $T_{sb}(o)$ and $S_{sb}(o)$.

Sometime

A sometime preference has the following PDDL syntax $(sometime\ \Phi)$ where the formula Φ has to become true at least once state in the plan state trajectory.

Definition 11 *Given an operator o and a sometime preference $P = (sometime\ \Phi)$ of a STRIPS+P problem, o is a **potential satisfying** operator for P if o can make true Φ. If an operator o can not make true Φ then the operator is **neutral** for P.*

The set of sometime preferences of Π which are potentially satisfied by the operator o are denoted with $S_{st}(o)$.

At-most-once

An at-most-once preference has the following PDDL syntax $(at-most-once\ \Phi)$ where the formula Φ has to become true at most once in the plan state trajectory.

Definition 12 *Given an operator o and an at-most-once preference $P = (at-most-once\ \Phi)$ of a STRIPS+P problem, o is a **potential threat** operator for P if o could make true Φ.*

We distinguish the following situations:

- if Φ has never become true in states earlier than the state s where o is applied and Φ becomes true in the state resulting from the application of o in s, then the corrispondent compiled operator o' has to take account this fact, otherwise, if Φ has become true in a earlier state, then o is a *violation*;
- if Φ does not become true in the state resulting from the application of o, then o is a *neutral* operator.

The set of at-most-once preferences of Π which are potentially threatned by the operator o are denoted with $T_{amo}(o)$.

Compilation intro STRIPS+

Definition 13 *If an operator $o \in O$ is safe for every always preference in P and neutral for everey sometime-before, at-most-once and sometime preference in P then we say that o is **neutral** for the problem Π and we write this property with $neutral(o)$. The set containing all the neutral operators for Π is defined as $N(\Pi) = \{o \in O \mid neutral(o)\}$.*

Definition 14 *Given an operator $o \in O$ of a STRIPS+P problem Π $I(op)$ is the following set: $\{T_a(o) \cup T_{sb}(o) \cup S_{sb}(o) \cup T_{amo}(o) \cup S_{st}(o)\}$ which contains all the preferences $p \in P$ of Π which are affected by the execution of o.*

Given a STRIPS+P problem, an equivalent STRIPS+ problem can be derived by translation which has some similarities to what proposed by Keyder and Geffner for soft goals but also significant difference. The scheme proposed by Keyder and Geffner is considerable simpler than ours because it does not to consider the interaction between actions and preferences such as threats, supports and violations. In order to simplify the compilation scheme we don't consider the compilation of soft goals because it can be easily added using the same method of Keyder and Geffner. Moreover we assume that every always and sometime-before preference is satisfied in the problem initial state I.

Given a STRIPS+P problem $\Pi = \langle F, I, O, G, P, c, u\rangle$, the compiled STRIPS+ problem of Π is $\Pi' = \langle F,' I', O', G', P', c'\rangle$ with:

- $F = F' \cup V_{a,sb,st,amo} \cup D \cup C \cup \overline{C'} \cup \{normal\text{-}mode, end\text{-}mode, pause\}$;
- $I' = I \cup \overline{C'}_{st} \cup V_{st} \cup \{normal\text{-}mode\}$;
- $G' = G \cup C'$;
- $O' = \{collect(st), forgo(st) \mid st \in ST \subseteq P\} \cup \{end\} \cup O_{comp}$
- $c'(o) = \begin{cases} u(st) & \text{if } if o = forgo(st), st \in ST \\ c(o) & \text{if } o \in N(\Pi) \\ c_{tv}(o) & \text{if } o \notin N(\Pi) \\ 0 & \text{otherwise} \end{cases}$

where:

- $V_{a,sb,st,amo} = \cup_{i=1}^{k}\{P_i\text{-}violated\}, k = |P|$;
- $D = \cup_{i=1}^{n}\{P_i\text{-}done_{o_1}, ..., P_i\text{-}done_m\}$ where $n = |O|$ and $m = |I(o)|$;
- $C'_{st} = \{ST'_i \mid ST_i \in ST \subseteq P\}$ and $\overline{C'}_{st} = \{\overline{ST'_i} \mid ST_i \in ST \subseteq P\}$
- $V_{st} \subseteq V_{a,sb,st,amo}$;
- $collect(ST_i) = \langle\{end\text{-}mode, \neg ST_i\text{-}violated, \overline{ST'_i}\}, \{ST', \neg\overline{ST'}\}\rangle$
- $forgo(ST_i) = \langle\{end\text{-}mode, ST_i\text{-}violated, \overline{ST'_i}\}, \{ST', \neg\overline{ST'}\}\rangle$
- $end = \langle\{normal\text{-}mode, \neg pause\}, \{end\text{-}mode, normal\text{-}mode\}\rangle$
- $O_{comp} = O_{neutral} \cup O_{chained} \cup O_{violation}$
- $O_{neutral} = \{\langle pre(o) \cup \{normal\text{-}mode, \neg pause\}, eff(o)\}\rangle \mid o \in O \wedge o \in N(\Pi)$;
- $O_{chained}$ and $O_{violation}$ are the compiled operators sets generated by the transformation schema applied to the operators of Π that threaten, violate or interact with at least a

preference of Π. An operator $o \in O$ is compiled through the compilation schema if $|I(o)| > 0$; the compiled operators $o_{chained}$ of the non-neutral operators are defined as: $\bigcup_{o \in O, o \notin N(\Pi)} chain(o)$ where $chain(o)$ is a function defined further down;

- $c_{tv}(o)$ is the cost of an operator $o \notin N(\Pi)$.

For each sometime preference ST, the transformation of Π into Π' adds a dummy hard goal ST' to Π' which can be achieved in two ways: with action $collect(ST)$, that has a cost 0 but requires that ST is satisfied, or with action $forgo(ST)$, that has a cost equal to utility of ST and can be performed when ST is unsatisfied in s_n. Note that the original initial state I is extended with the V_{st} set, which contains, for each $ST \in STS \subseteq \mathcal{P}$, a literal $is\text{-}violated\text{-}ST$ stating that ST is violated until a $o \in S_{st}(o)$ satisfies the associated formula. For each sometime preference exactly one of $\{collect(ST), forgo(ST)\}$ appears in the plan.
This approach is not used for every other kind of preference, except sometime, whose violation is catched by the model during planning and not at the end of the planning.

The compilation schema

Each operator o such that $|I(o)| > 0$, or equivalently $o \notin N(\Pi)$, is compiled into a set of new operators. The set of the m preferences affected by o is $I(o) = \{P_1, ..., P_m\}$. Then o is compiled into a set of operators $chain(o) = \{\Theta(o, P_1), ..., \Theta(o, P_m)\}$ where each $\Theta(o, P_i)$ for $i \in [1, ..., m]$ is a set of operators, called *stage*, related to an affected preference $P_i \in I(o)$. The definition of each stage $\Theta(o, P_i)$ depends on the kind of preference $typ(P_i)$ and the value of i. Furthermore the stage set are built in order to execute the following operators sequence $\omega_{chain(o)} = \langle o'_{P_1}, ..., o'_{P_m} \rangle$ where o'_{P_i}, with $i \in [1, ...m]$, is selected from the i-th set $\Theta(o, P_i)$.

Given a not-neutral operator o of Π, the set of the compiled operators related to o for Π', called chain for o, is defined as:

$$chain(o) = \bigcup_{p_i \in I(o), i \in [1,...,|I(o)|]} \Theta(o, p_i)$$

This set is called *chain* because the operators in each stage are built in order to force the sequential execution of $\omega_{chain(o)}$.

In this presentation I provide the detailed descripion for the compilation of an operator o that affects the i-th at-most-once preference in $I(o)$.

Definition 15 *The compilation-method for the translation of a non-neutral operators o that affect the i-th at-most-once preference* $P_i = (\text{at-most-once } a_i)$ *where* $a_i = \bigwedge_j a_{ij}$ *(where* a_{ij} *is a clause) of* $I(o)$ *is:*

- *if* $i = 1$ *(init stage):*
  $prec(o_{a_1}) = prec(o) \cup \{\neg pause, \neg is\text{-}violated\text{-}a_1, \neg seen\text{-}a_1\} \cup \{\bigcup_{a_{1j} \in \overline{C}(o,a_i)} a_{1j}\}$
  $eff(o_{a_1}) = \{pause, seen\text{-}a_1, a_1\text{-}done_o\}$

  $prec(\overline{o}_{a_1}) = prec(o) \cup \{\neg pause, \neg is\text{-}violated\text{-}a_1\} \cup \neg\{\bigwedge a_{1j} \in \overline{C}(o, a_i) a_{1j}\}$
  $eff(\overline{o}_{a_1}) = \{pause, a_1\text{-}done_o\}$

  $prec\left(\overline{\overline{o}}_{a_1}\right) = prec(o) \cup \{\neg pause, is\text{-}violated\text{-}a_1\}$
  $eff\left(\overline{\overline{o}}_{a_1}\right) = \{pause, a_1\text{-}done_o\}$

- *if* $1 < i < m = |I(o)|$ *(middle stage):*
  $prec(o_{a_i}) = \{pause, \neg is\text{-}violated\text{-}a_i, \neg seen\text{-}a_i, a_{i-1}\text{-}done_o\} \cup \{\bigcup_{a_{ij} \in \overline{C}(o,a_i)} a_{ij}\}$
  $eff(o_{a_i}) = \{pause, seen\text{-}a_i, \neg a_{i-1}\text{-}done_o, a_i\text{-}done_o\}$

  $prec(\overline{o}_{a_i}) = \{pause, \neg is\text{-}violated\text{-}a_i, seen\text{-}a_i, a_{i-1}\text{-}done_o\} \cup \{\bigcup_{a_{ij} \in \overline{C}(o,a_i)} a_{ij}\}$
  $eff(\overline{o}_{a_i}) = \{pause, is\text{-}violated\text{-}a_i, \neg a_{i-1}\text{-}done_o, a_i\text{-}done_o\}$

  $prec(\overline{\overline{o}}_{a_i}) = \{pause, \neg is\text{-}violated\text{-}a_i, a_{i-1}\text{-}done_o\} \cup \neg\{\bigwedge a_{ij} \in \overline{C}(o, a_i) a_{ij}\}$
  $eff(\overline{\overline{o}}_{a_i}) = \{pause, \neg a_{i-1}\text{-}done_o, a_i\text{-}done_o\}$

  $prec\left(\overline{\overline{\overline{o}}}_{a_i}\right) = \{pause, is\text{-}violated\text{-}a_i, a_{i-1}\text{-}done_o\}$
  $eff\left(\overline{\overline{\overline{o}}}_{a_i}\right) = \{pause, a_{i-1}\text{-}done_o, \neg a_{i-1}\text{-}done_o, a_i\text{-}done_o\}$

- *if* $i = m$ *(final stage):*
  $prec(o_{a_m}) = \{pause, \neg is\text{-}violated\text{-}a_m, \neg seen\text{-}a_m, a_{m-1}\text{-}done_o\} \cup \{\bigcup_{a_{mj} \in \overline{C}(o,a_m)} a_{mj}\}$
  $eff(o_{a_m}) = eff(o) \cup \{\neg pause, seen\text{-}a_m, \neg a_{m-1}\text{-}done_o\}$

  $prec(\overline{o}_{a_m}) = \{pause, \neg is\text{-}violated\text{-}a_m, seen\text{-}a_m, a_{m-1}\text{-}done_o\} \cup \{\bigcup_{a_{mj} \in \overline{C}(o,a_m)} a_{mj}\}$
  $eff(\overline{o}_{a_m}) = eff(o) \cup \{\neg pause, is\text{-}violated\text{-}a_m, \neg a_{m-1}\text{-}done_o\}$

  $prec(\overline{\overline{o}}_{a_m}) = \{pause, \neg is\text{-}violated\text{-}a_m, a_{m-1}\text{-}done_o\} \cup \neg\{\bigwedge a_{mj} \in \overline{C}(o, a_m) a_{mj}\}$
  $eff(\overline{\overline{o}}_{a_m}) = eff(o) \cup \{\neg pause, \neg a_{m-1}\text{-}done_o\}$

  $prec\left(\overline{\overline{\overline{o}}}_{a_m}\right) = \{pause, is\text{-}violated\text{-}a_m, a_{m-1}\text{-}done_o\}$
  $eff\left(\overline{\overline{\overline{o}}}_{a_m}\right) = eff(o) \cup \{\neg pause, a_{m-1}\text{-}done_o, \neg a_{m-1}\text{-}done_o\}$

In accordance with Definition 12 the i-th stage $\theta_{amo}(o, P_i)$ providing the following possible choices:

- o_{a_i} is a *neutral* operator for P_i which asserting that the related formula a_i has been seen for the first time;

- $\overline{o}_{a_i}$ is a *violation* of P_i because the related formula a_i has been true in a previous state;
- $\overline{\overline{o}}_{a_i}$ is a *neutral* operator for P_i because it does not make a_i true;
- $\overline{\overline{\overline{o}}}_{a_i}$ is a *neutral* operator for P_i because it has already been violated in a previous state.

Conclusion

In my first years of PhD, I have worked on the compilation of PDDL 3.0 preferences into STRIPS with action costs. As a base I started from two works of (Keyder and Geffner 2009), for the compilation of soft goal, and (Ceriani and Gerevini 2015) for the compilation of always goal. I have developed a new compilative scheme for three type of preference which were not considered in the previous work. All the propose compilative methods have been implemented and preliminary experiments show that the investigated approach is competitive in terms of performance with other existing approaches to planning preferences.

References

Ceriani, L., and Gerevini, A. E. 2015. Planning with always preferences by compilation into strips with action costs. In *Eighth Annual Symposium on Combinatorial Search*.

Gerevini, A. E.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5):619–668.

Keyder, E., and Geffner, H. 2009. Soft goals can be compiled away. *Journal of Artificial Intelligence Research* 547–556.

Planning Under Uncertainty with Temporally Extended Goals

Alberto Camacho *
Department of Computer Science
University of Toronto. Canada.
acamacho@cs.toronto.edu

1 Introduction

In the last decade, we have seen an exponential increase in the number of devices connected to the Internet, with a commensurate explosion in the availability of data. New applications such as those related to smart cities exemplify the need for principled techniques for automated intelligent decision making based on available data. Many decision-making problems require reasoning in large and complex state spaces, sometimes under stringent time constraints. The nature of these problems suggests that planning approaches could be used to find solutions efficiently. Automated planning is the basis for a diversity of problems such as automated diagnosis, controller synthesis, and story generation and understanding. Nevertheless, most studied planning models make assumptions that do not hold in many real-world problems. These include assumptions regarding the nature of actions (e.g. assumed to be deterministic) and goals (e.g. assumed to pertain only to the final state).

We are interested in exploring and formalizing new planning models that capture properties of existing real-world problems with the aim of moving a step forward towards the design of efficient and scalable algorithms. To date, our contributions in FOND and probabilistic planning provide a foundation for future exploration and advances with more complex models, including those with continuous variables, and temporal planning on the horizon.

Illustrative Example Consider the problem of designing a tourist route to visit a set of touristic attractions in London. The tour is subject to certain constraints, such as visiting places in a specific order, not visiting the London Eye before the Houses of Parliament, and visiting the Maritime Museum right after the Greenwich Observatory. Moreover, plans need to include a safety constraint to ensure the tourist is drinking water every hour. These are examples of temporally extended goals. Following with our example, certain dynamics of the environment are not controllable to the agent, such as traffic, punctuality of public transport, and the weather. If the stochastic model for these events is available, we can quantify the *expected quality* of the plan according to a certain metric (e.g. probability of visiting at least 5 touristic attractions at the end of the journey) and attempt to produce plans that maximize it. When the stochastic model is not available, we may want to produce plans that are robust to *any* incontingency (e.g. a plan that suggests visiting museums, at any moment, if it starts raining).

*The contributions presented in this paper reflect joint work with (in alphabetical order) Jorge Baier (jabaier@ing.puc.cl), Sheila McIlraith (sheila@cs.toronto.edu), Christian Muise (cjmuise@mit.edu), and Eleni Triantafilou (eleni@cs.toronto.edu).

2 Progress to the Date

In our work to day, we have advanced the state of the art in planning problems with non-deterministic actions and temporally extended goals. In this section, we introduce the FOND and probabilistic planning models, and describe the high-level contributions of our work. We refer the reader to the respective publications for further details.

A *Fully Observable Non-Deterministic* (FOND) planning problem is a tuple $\mathcal{P} = \langle S, s_I, \mathcal{A}, F, S_\mathcal{G} \rangle$, where S is a finite set of *states*, $s_\mathcal{I} \in S$ is the *initial state*, $S_\mathcal{G} \subseteq S$ is a set of *goal states*, and $\mathcal{A}$ is a finite set of actions. For each action $a \in \mathcal{A}$, and state $s \in S$, the result of applying a in s is one of the states in the set $F(s, a) \subseteq S$. Solutions to FOND planning problems are *policies*, or mappings $\pi : S \to \mathcal{A}$ from states into actions. In concrete, *strong-cyclic* solutions are those that lead the agent to a goal state with complete guarantees (Cimatti et al. 2003).

A *probabilistic* planning problem is a tuple $\mathcal{P} = \langle S, s_I, \mathcal{A}, T, S_\mathcal{G} \rangle$. Different than the FOND model, for each action $a \in \mathcal{A}$, and pair of states $s, s' \in S$ the $T(s, a, s')$ is the *transition probability* of reaching s' when a is applied in s. Solutions to probabilistic planning problems are *policies*. In goal-oriented probabilistic planning models such as MaxProb, solutions are policies that lead the agent to a goal state with maximal probability.

2.1 ProbPRP

In (Camacho, Muise, and McIlraith 2016) we present ProbPRP, a probabilistic planner that finds solutions to probabilistic planning problems where the objective is to attempt to maximize the probability of reaching a goal state. We formalize this class of problems and call it HighProb.

ProbPRP has two important merits. First, it overpasses difficulties that previous offline algorithms experienced to

scale in big problems. And second, it offers increased optimality guarantees with respect to the previous state of the art in HighProb, the online planner RFF (Teichteil-Königsbuch, Kuter, and Infantes 2010). Despite being an offline algorithm, ProbPRP outperforms RFF in general and solutions are of better quality.

ProbPRP leverages core similarities between probabilistic and FOND planning, making use of state-of-the-art FOND planning techniques from PRP (Muise, McIlraith, and Beck 2012) in its underlying algorithm. The *partial state* representation obtained via plan regression facilitates states entailment during the search process, and results in considerable improvements in the algorithm convergence. Besides, the compact representation of state results in smaller policies. The *deadend detection* mechanism prunes the search space effectively by means of forbidden state-action pairs (FSAPs) generated automatically during the search process, and guarantees optimality of the algorithm when deadends are avoidable.

ProbPRP extends the state-of-the-art FOND planner PRP (Muise, McIlraith, and Beck 2012) with techniques that leverage probabilistic information to produce high quality solutions. Some of these enhancements of ProbPRP include the bias towards search of *high-likelihood plans*, and the *final FSAP-free round*. ProbPRP biases search towards exploration of high-likelihood plans. As result, policies have smaller expected plan length, which is orders of magnitude lower in the most beneficial cases. A final search round is performed to extend the best incumbent policy found by the algorithm, this time with the FSAP mechanism disabled. The probability of reaching a goal state of the final policies increment up to 30% in the most beneficial cases.

2.2 LTL FOND Translations

In (Camacho et al. 2016) we address the problem of planning with non-deterministic actions and temporally extended goals. We assume goals are specified as LTL formulas (Pnueli 1977), and call the model LTL FOND.

LTL formulae can be interpreted over finite or infinite state trajectories. Solutions to different interpretations are not always equivalent. A number of techniques exist to solve planning with LTL goals, with and without presence of non-deterministic actions, and with finite and infinite LTL interpretations. A common approach is to compile the problem into one with final-state goal, and solve the resulting problem with state-of-the-art planning technology (e.g. (Baier and McIlraith 2006; Patrizi, Lipovetzky, and Geffner 2013; Torres and Baier 2015)).

We present translations to compile LTL FOND into FOND. In concrete, two translations that handle finite LTL interpretations, and two translations that handle infinite LTL interpretations. Remarkably, we are the first to solve the full spectrum of LTL FOND planning interpreted on inifite state trajectories. Equipped with strong-cyclic planner, PRP, our system proves competitive with other state-of-the-art algorithms for LTL FOND, with the advantage of being able to solve the full spectrum of LTL FOND problems.

Our translations leverage ideas from (Baier and McIlraith 2006; Torres and Baier 2015; Patrizi, Lipovetzky, and Geffner 2013), and use Non-deterministic Finite Automata (NFA) and Alternating Automata (AA) representations of the LTL formula to monitor progression, and strong-cyclic planning to synthesize solutions. The size of NFA-based translations is worst-case exponential in the size of the formula, and the size of AA-based translations is worst-case polynomial. Interestingly, PRP performance was better with NFA-based translations, with smaller policies and lower runtimes than with AA-based translations.

3 Future Work

A natural next step is to extend our recent work by defining the class of probabilistic planning problems with LTL goals and solving them using our existing translations and ProbPRP as a probabilistic planner. This raises a more general question, which is to characterize the class of planning problems with LTL goals in which our translations are applicable.

We plan to explore other planning models that capture further properties reflected in real-world problems. This includes studying a more general description of actions and variables, e.g. stochastic actions, planning with continuous variables, temporal planning, and hybrid planning.

References

Baier, J., and McIlraith, S. 2006. Planning with first-order temporally extended goals using heuristic search. In *AAAI*, 788–795.

Camacho, A.; Triantafilou, E.; Baier, J. A.; Muise, C.; and McIlraith, S. A. 2016. LTL Synthesis for Non-Deterministic Systems on Finite and Infinite Traces. Under Review.

Camacho, A.; Muise, C.; and McIlraith, S. A. 2016. From fond to robust probabilistic planning: Computing compact policies that bypass avoidable deadends. In *ICAPS*.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *AIJ* 147:35–84.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2012. Improved Non-deterministic Planning by Exploiting State Relevance. In *ICAPS*, 172–180.

Patrizi, F.; Lipovetzky, N.; and Geffner, H. 2013. Fair LTL synthesis for non-deterministic systems using strong cyclic planners. In *IJCAI*.

Pnueli, A. 1977. The temporal logic of programs. In *FOCS*, 46–57.

Teichteil-Königsbuch, F.; Kuter, U.; and Infantes, G. 2010. Incremental plan aggregation for generating policies in MDPs. In *AAMAS*, volume 1, 1231–1238.

Torres, J., and Baier, J. A. 2015. Polynomial-time reformulations of LTL temporally extended goals into final-state goals. In *IJCAI*, 1696–1703.

Temporal Inference In Forward Search Temporal Planning Dissertation Abstract

Atif Talukdar
Supervisors: Maria Fox and Derek Long
King's College London
London WC2R 2LS
atif.talukdar@kcl.ac.uk

Abstract

Forward search planners are typically good at using search mechanisms to find solutions to a problem. They do search by performing state evaluations and selecting a promising successor state to navigate to. These planners often use a relaxation to solve an easier form of the problem in each state evaluation, in order to attain heuristic guidance. A popular relaxation is the delete relaxation used by planners such as FF (Hoffmann and Nebel 2001). However, forward search planners can often find it difficult to make use of inference, and as a result lose the power that inference brings. Without inference, a planner needs to search for every action, even ones that could have been inferred without any search. This paper identifies patterns of required concurrency and provides an analysis of these patterns and the inferences that can be deduced from them. We discuss ideas of how these inferences can be implemented in a current forward search planner, POPF (Coles et al. 2010).

1 Introduction

In temporal planning, an interesting class of problems are those which require close temporal coordination between actions within a finite time window. Some of these interactions arise where actions need to occur concurrently in order for a solution to be found. Required concurrency is where two or more actions must occur at least partly within the duration of the other, for all possible plans to the problem (Cushing et al. 2007).

Currently, even powerful forward search temporal planners still perform search for required actions that could be inferred. However, search is still needed since the required concurrency is not explicitly identified. Recognising the types of situations where required concurrency exists, and identifying the patterns in which they occur, provides valuable information. It is possible to leverage this information within a forward search planning framework, to solve problems with required concurrency, faster and with less search. The current work focusses on required concurrency between two actions only.

The patterns of required concurrency presented in this paper, are based on those identified by Cushing et al. (2007). However, the types of required concurrency they identify are in the context of grounded actions, with specific problem goals identified. Our work currently abstracts from grounded actions in the detection of required concurrency, to give an analysis of the action definitions in the domain structure, before grounding takes place. Our objective is to automate the recognition of these patterns in the domain structure before the planner begins its search process. This can be used to identify opportunities for using inference during plan construction instead of pure search. If one action in a required concurrency pair is added to the plan, and the other is not already in the plan, then it can be added through inference, reducing the amount of search. In addition, for problems with required concurrency, states that do not have the required actions occurring concurrently, can be pruned from the search space.

We recognise that in constraint based planners such as CPT (Vidal and Geffner 2004) and eCPT (Vidal and Geffner 2005), inference is exploited as a powerful pruning tool. Our intention is to explore how inference can be used in a forward-chaining search framework, where a plan head is maintained at each state. We intend to compare and evaluate the benefits of inference between the two approaches at a later stage, once our approach has been fully implemented and tested.

In a similar way to how POPF provides a compromise between the principles of least commitment in partial ordering and total ordering commonly used in forward search, we propose a compromise between search using relaxed heuristic guidance and inference. Using inference in forward search has potentially immense benefits, when the action added via search that triggers an inference is the right choice. If the action that triggers an inference is found to have been a wrong choice later on during plan construction, the impact of backtracking is reduced by the fact that some actions were added without an expensive search process. Presently, we focus our analysis on propositional domain problems, with the intent to scale up as we develop and test our algorithms.

Section 2 describes the patterns of required concurrency. Section 3 describes the inferences possible from each pattern of required concurrency and the actions needed in the

plan to form these inferences. Section 4 describes how we intend to use inference in a forward chaining search framework to reduce search. Section 5 discusses the current state of the implementation of our approaches in POPF. Section 6 concludes the report and outlines the next stage of work.

2 Patterns of Required Concurrency and Temporal Constraints

This section discusses the patterns of required concurrency between two actions which are identified using a pre-search analysis of the plan head, and our method of representing the various constraint types. The Simple Temporal Network (STN) diagrams represent the temporal constraints between actions that must exist by the end of plan construction, if those actions are used in the plan. Each pattern is identified in the top sub-figure, and its corresponding STN in the bottom sub-figure. For the pattern diagrams, the letters above the actions represent the start, overall, and end preconditions going from left to right. The letters below are the start and end effects going from left to right. Diagrams illustrating the patterns of required concurrency in figures 2 to 8 do not display action durations. We assume that in patterns identifying cases where an action B, needs to occur entirely or partly within the duration of another action A, that the duration of A must be long enough to encompass that part of action B.

2.1 STN Constraint Types

We describe three types of ordering relation which are maintained in the STNs. The block arrows in the STN diagrams shown in figure 1a, represent the constraint that any start action, denoted $A_{\vdash}$, must be followed by its corresponding end action, denoted $A_{\dashv}$. This is the relationship between start and end snap actions as presented in (Coles et al. 2008). This Start-End relationship is already maintained in the STN by POPF. The single solid line arrows with block heads illustrated in figure 1b represent the contingent constraint between a concurrent pair of actions, showing that both must be in the plan head with the action being pointed to, occurring after the action being pointed from. Actions connected by a contingent constraint must be in the plan head in the order shown, for the inference to take place. The broken line arrows with a hollow head, shown in figure 1c depict the constraints which are inferred from knowing the start-end and contingent constraints, using their respective arrow types.

(a) Start-End Constraint.

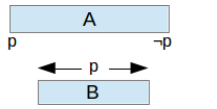
(b) Contingent Constraint.

(c) Inferred Constraint.

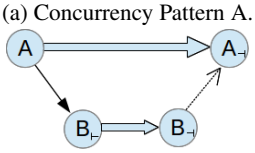
Figure 1: Types of constraints between actions in STNs.

2.2 Patterns of Required Concurrency

We start by noting that for the current work, we assume that there is a single achiever action for each fact that appears in a pattern of required concurrency. Concurrency pattern A in figure 2a displays the situation where one action A, provides a resource "p", for its duration only. Any action B, which requires this resource must occur within the temporal window created by action A. In this case action B requires resource "p" for its entire duration, which is provided by A; therefore action B must occur entirely within the execution of A. As soon as the plan head contains $A_{\vdash}$ and $B_{\vdash}$, with $A_{\vdash}$ before $B_{\vdash}$, it can be inferred that $B_{\dashv} < A_{\dashv}$.



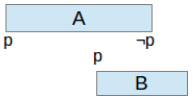
(a) Concurrency Pattern A.



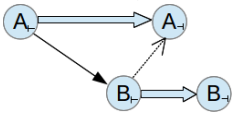
(b) STN Constraints.

Figure 2: Concurrency Pattern A and Inferred Temporal Constraints.

Concurrency pattern B in figure 3a shows the same temporal window created by A for resource P, however in this case, B only needs the effect of A as a start precondition, therefore $B_{\vdash}$ must occur after $A_{\vdash}$ and before $A_{\dashv}$, but $B_{\dashv}$ can come after $A_{\dashv}$. As soon as $A_{\vdash}$ and $B_{\vdash}$ appear in the plan head, it can be inferred that $B_{\vdash} < A_{\dashv}$.



(a) Concurrency Pattern B



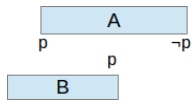
(b) STN Constraints

Figure 3: Concurrency Pattern B and Inferred Temporal Constraints.

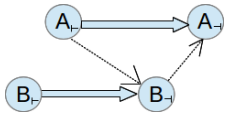
Concurrency pattern C in figure 4a is the same situation again, except that action B requires P as an end precondition. However, we can do more inference in pattern C than in patterns A and B. This is since, as soon as $B_{\vdash}$ is in the plan head, we can infer that $A_{\vdash} < B_{\dashv}$ and $B_{\dashv} < A_{\dashv}$.

We can see that patterns A and B are more constrained that pattern C, in terms of the power of the inference. This is since pattern A and B required the start of both actions A and B to be in the plan head, to trigger the inferred ordering

constraints. Furthermore, any other actions conflicting with B, that also need to occur within the envelope of A, would result in tighter scheduling being required between the actions.



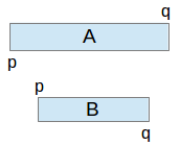
(a) Concurrency Pattern C



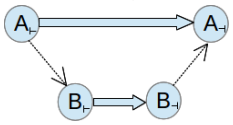
(b) STN Constraints

Figure 4: Concurrency Pattern C and Inferred Temporal Constraints.

Concurrency pattern D in figure 5a represents another scenario where one action must occur entirely within the execution of another action, similar to the situation of concurrency pattern A in figure 2a. However, in this case the reasoning is different, since action A does not create a temporal window for the availability of a resource. Instead action B requires the effect of $A_{\vdash}$, which persists following $A_{\dashv}$. Therefore $B_{\vdash}$ must come after $A_{\vdash}$. However, $B_{\dashv}$ has an effect which is the end precondition for $A_{\dashv}$, thus $B_{\dashv}$ must also come before $A_{\dashv}$. Each action produces an effect which the other action needs as a precondition. In the case of pattern D, the effect of each action is needed as a precondition of the other at the same end point. Pattern D is recognised as soon as $A_{\vdash}$ appears in the plan head. This inference is quite powerful, as we are able to immediately commit $A_{\vdash} < B_{\vdash}$ and $B_{\dashv} < A_{\dashv}$.



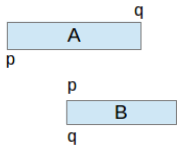
(a) Concurrency Pattern D.



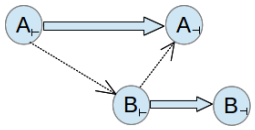
(b) STN Constraints.

Figure 5: Concurrency Pattern D and Inferred Temporal Constraints.

Concurrency pattern E in figure 6a displays a similar situation to pattern D, except the effect that B provides, needed by $A_{\dashv}$, is now provided by $B_{\vdash}$, instead of $B_{\dashv}$. This in effect produces the same type of required concurrency as pattern B, however there is again a different reason for it. $B_{\vdash}$ must occur after $A_{\vdash}$ and before $A_{\dashv}$, but $B_{\dashv}$ can occur after $A_{\dashv}$. This is because the precondition of $A_{\dashv}$ is this time produced by $B_{\vdash}$ instead of $B_{\dashv}$. Again, there is a powerful inference available since the appearance of $A_{\vdash}$ in the plan head allows us to infer $A_{\vdash} < B_{\vdash}$ and $B_{\vdash} < A_{\dashv}$.



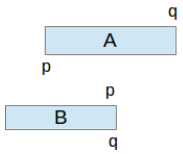
(a) Concurrency Pattern E.



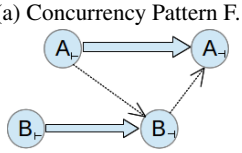
(b) STN Constraints.

Figure 6: Concurrency Pattern E and Inferred Temporal Constraints.

Concurrency pattern F in figure 7a presents a similar situation as pattern C, except that fact "p" is needed by B as an end precondition and provides "q" as its end effect, which action A needs as its end precondition. The temporal constraints in the STN of pattern F in figure 7b can be deduced after the addition of either $A_{\vdash}$ or $B_{\vdash}$ to the plan. As soon as $A_{\vdash}$ or $B_{\vdash}$ appears in the plan head, we can infer $A_{\vdash} < B_{\dashv}$ and $B_{\dashv} < A_{\dashv}$.



(a) Concurrency Pattern F.

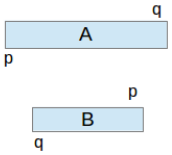


(b) STN Constraints.

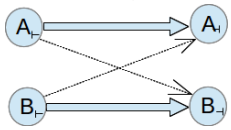
Figure 7: Concurrency Pattern F and Inferred Temporal Constraints.

Concurrency pattern G presents the case where an end precondition of each action in the pair is provided by the start effect of the other. For this reason, the minimal amount of required concurrency between a pair of actions of this pattern, is where only one end point of both actions, must occur during the execution of the other. The most optimal form of concurrency, in regards to plan makespan, being where one action is executed entirely during the execution of the other. As soon as $A_{\vdash}$ or $B_{\vdash}$ appears in the plan head, we can infer $A_{\vdash} < B_{\dashv}$ and $B_{\vdash} < A_{\dashv}$.

Patterns F and G are the most powerful of all, since inference can be made based on the appearance of either $A_{\vdash}$ or



(a) Concurrency Pattern G.



(b) STN Constraints.

Figure 8: Concurrency Pattern G and Inferred Temporal Constraints.

$B_{\vdash}$ in the plan head. In all patterns A to G, as soon as the available inferences are made, the temporal constraints can be added to the plan without search.

3 Inferences from Concurrency Patterns

This section describes the inferences possible from each pattern of required concurrency. Table 1 displays which actions need to be in the plan as input, to infer the corresponding temporal constraints from each pattern of required concurrency as the output. Figure 9 displays the increase in the power of inference from the patterns of required concurrency. The inference is very powerful in patterns C to G, since the start of one action in the plan, allows us to infer that another action must be added to the plan, to satisfy all the preconditions. This is additional information we can infer compared to patterns A and B, where actions $A_{\vdash}$ and $B_{\vdash}$ are already in the plan, and only the ordering of the action ends can be inferred.

The constraints which are inferred through transitivity, given the actions already in the plan head are shown in brackets in table 1. Figure 9 show patterns A and B to be in the first level, since only ordering constraints can be inferred; actions $A_{\vdash}$ and $B_{\vdash}$ must already be in the plan head. Although patterns C to G all allow a new action to be inferred as required in the plan, they have been split into another two levels. This is since patterns C, D, E require a specific action to infer their concurrent action. In contrast, patterns F and G are more flexible in that the appearance of either action in the pair, allows the inclusion of other to be inferred, making them more powerful.

4 Using Inference to Reduce Search

In this section, we discuss how inference reduces search and our approach to perform the required concurrency detections and inferences. Forward search planners typically navigate the search space, by evaluating potential successor states from the current state. This can often result in a large number of state expansions, even in cases there is required concurrency and only a single solution to the problem exists. We propose to reduce search in the forward plan

Pattern	Current Plan	Constraints Inferred
A	$A_{\vdash} < B_{\vdash}$	$B_{\dashv} < A_{\dashv}$
		$(B_{\vdash} < A_{\dashv})$
B	$A_{\vdash} < B_{\vdash}$	$B_{\vdash} < A_{\dashv}$
C	$B_{\vdash}$	$A_{\vdash} < B_{\dashv}$
		$B_{\dashv} < A_{\dashv}$
		$(B_{\vdash} < A_{\dashv})$
D	$A_{\vdash}$	$A_{\vdash} < B_{\vdash}$
		$B_{\dashv} < A_{\dashv}$
		$(B_{\vdash} < A_{\dashv})$
E	$A_{\vdash}$	$A_{\vdash} < B_{\vdash}$
		$B_{\vdash} < A_{\dashv}$
F	$A_{\vdash} \vee B_{\vdash}$	$A_{\vdash} < B_{\dashv}$
		$B_{\dashv} < A_{\dashv}$
		$(B_{\vdash} < A_{\dashv})$
G	$A_{\vdash} \vee B_{\vdash}$	$A_{\vdash} < B_{\dashv}$
		$B_{\vdash} < A_{\dashv}$

Table 1: Inferred temporal constraints.

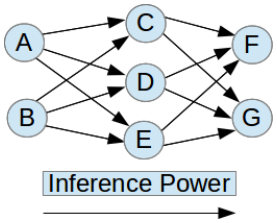


Figure 9: Increase in Power of Inference.

construction process, by detecting triggers for inference, and immediately adding the inferred actions to the plan, instead of doing state evaluation and heuristic search. This type of inference applies to concurrency pairs for patterns of types C to G. A trigger action is one that identifies that its partner action must be added and occur concurrently. Our detection process identifies pairs of concurrent actions within the domain at a pre-search stage before any plan construction begins. If an action is added to the plan via the normal search process, and is identified as a trigger action, its corresponding partner action will be added to the plan instead evaluating the successor state and searching for the next action.

The example in figure 10 displays a situation where a chain of inferences are made, allowing for a large reduction in the number of state evaluations. Suppose we have a problem with an empty initial state and a goal state with fact “g”. Although Action_1 achieves the goal, the plan requires all the actions displayed in figure 10, due to the interdependencies between the actions. Currently it takes POPF 20 state evaluations to reach a solution using the existing search machinery. Using our proposed approach for inference, it is possible to reduce this to a single state evaluation. We can see that Action_1 achieves the goal fact and that the other actions are needed, each as a partner action in a concurrency pattern. According to the patterns of required concurrency

identified in section 2, there are 4 patterns identified in figure 10, D, E, F and G. Action_1 is a trigger action for pattern D, causing the addition of Action_2, this is a trigger for a pattern E pair, adding Action_3. Action_3 is then the trigger for adding Action_4 in a pattern F pair. Lastly, either Action_4 or Action_5 is the trigger for inferring and adding the other, in a pattern G pair. Since either the start or end of each action satisfies at least one precondition of another, all of these actions are needed in the plan to reach a solution.

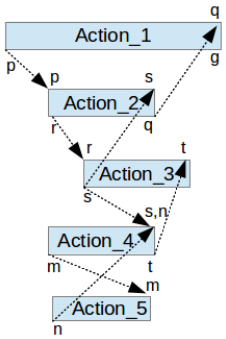


Figure 10: Chain of Inferred Actions. The arrows represent inferred ordering constraints. In this problem example, the initial state is empty and the goal state is to make fact "g" true.

5 Implementation

Here, we describe our progress in implementing our techniques for detecting required concurrency, doing inference and our process for interleaving search with inference .

5.1 Detecting Required Concurrency

The technique for detecting required concurrency works in two core parts. The first part works by searching through the domain structure for durative actions, and extracting relevant information from the condition and effects lists of the action definitions and storing possible candidates for required concurrency. The second part works by comparing the lists of candidate actions attached to each candidate predicate and matching action pairs which meet the criteria of a particular pattern of required concurrency. The algorithms for implementing both of these components have been implemented in the POPF code.

5.2 Inference Instead of Search

Any inference performed by the planner is triggered by a previous action added to the plan, either via search or through inference. Even if the trigger action is one added through inference like it is in figure 10 for a chain of inferences, the chain of actions will trace back to an action added via search. This ensures that actions added via inference are always required. Algorithm 1 displays our proposed method for deciding when the planner should use inference to add the next action in the plan, instead of using search.

An example of the standard search process currently used by POPF is shown in figure 11. Here, we see that S1 is evaluated and $A_{\vdash}$ selected, progressing the state to S2. State S2 is also then evaluated, providing 3 possible actions to apply next. Suppose that actions A and B make up a pattern D pair, applying A means that B must be applied concurrently. However, currently POPF still needs to evaluate S2 and choose either $B_{\vdash}$, $C_{\vdash}$ or $D_{\vdash}$ to progress the state to either S4, S5 or S6 respectively. If $B_{\vdash}$ is not selected due to one of the other actions having a better heuristic value, then $B_{\vdash}$ will still have to be in plan and therefore searched for, given that $A_{\vdash}$ has already been added to the plan.

We propose an alternative to this system, whereby if $A_{\vdash}$ is a trigger for inferring $B_{\vdash}$, instead of evaluating the successor state S2 and searching for the next action, we simply add $B_{\vdash}$ as the next action. This is provided that the preconditions of $B_{\vdash}$ are satisfied and it is applicable. Figure 12 shows the proposed approach of search and inference. Here, action $A_{\vdash}$ is a trigger for inference, therefore instead of evaluating the next state S2, we skip its evaluation, check that the inferred action $B_{\vdash}$ is applicable and if so, apply it straight away and progress to S4. This reduces search and the need to evaluate alternative actions, $C_{\vdash}$ and $D_{\vdash}$. Even if $C_{\vdash}$ and $D_{\vdash}$ are good choices to apply as the next action in S2, due to the detection of the required concurrency pattern, we know that $B_{\vdash}$ must go in the plan at some future point concurrently with $A_{\vdash}$. Therefore, if the inferred action is applicable and can be be applied next, we do so by promoting inference above search. If the inferred action $B_{\vdash}$ is not yet applicable, then it is stored, search is performed as normal and the applicability of $B_{\vdash}$ checked at the next successor state. If the inferred action never becomes applicable due to other unsatisfied preconditions, then the trigger action would be removed for action pairs of patterns D to G. However, if this were to happen, backtracking would be less expensive than having used pure search. The implementation of the search and infer process in algorithm 1 is currently in progress.

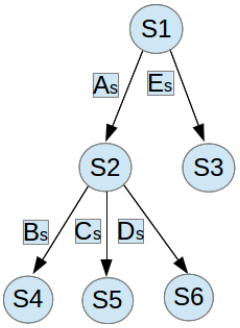


Figure 11: POPF using current search mechanism and no inference. S2 must be evaluated and alternative actions considered even though action B is part of a pattern of required concurrency.

Algorithm 1: searchAndInfer

Inout : States S, S', Actions $A\vdash$, $B\vdash$
evaluate S;
applyAction($A\vdash$ to S);
if *(S, $A\vdash$).triggersInference()* **then**
if *inferredAction $B\vdash$ isNotInPlan()* **then**
if *$B\vdash$.isApplicable()* **then**
  Recursively applyAction($B\vdash$ to S);
end
else
  store $B\vdash$;
  $S \leftarrow S'$;
  goto step 1;
end
else
  $S \leftarrow S'$;
  goto step 1;
end
else
  $S \leftarrow S'$;
  goto step 1;
end

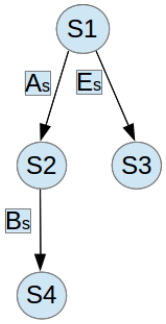


Figure 12: POPF detecting $A\vdash$ as a trigger for inference and applying $B\vdash$ without evaluating state S2, immediately progressing to S4.

6 Conclusion

The goal of this PhD is to exploit the power of inference in a forward search framework, which current forward search planners do not do. Our inferences are based on specific patterns of required concurrency detected in a pre-search analysis of the domain structure. Any action added via inference and not search, is an action that would have required search if inference had not been used. This is since detection of the patterns presented in section 2.2, guarantee that where one action of the pattern is in the plan, so must the other. Therefore, since an action that triggers an inference is added via the existing search mechanism, an action added via inference will always be a correct decision, given the actions already in the plan.

The next stage of work will be to complete the implementation of the search and inference mechanism outlined in algorithm 1. Once this is completed, a rigorous testing phase will be conducted to determine how robust the mechanisms for this search and infer process are and whether the results match our expectations. A good starting point will be to test if the 20 state evaluations currently needed to produce the plan of actions shown in figure 10, is reduced to a single state evaluation as expected. First, a robust implementation of the combined inference and search approach has to be achieved in domains with single achiever actions for each fact in a pattern. Following this, the next step will be to conduct research into required concurrency, where there are multiple achievers for these pattern facts. The challenge will be to identify which action instances to use for the required concurrency patterns, when the planner has a choice and how these choices can be prioritised. The work so far has focussed on the required orderings of concurrent actions, we do not currently use action durations for forming inferences. However, action durations are another aspect that we expect may be used as a foundation for further inferences in the future.

References

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with problems requiring temporal coordination. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, 892–897.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal? In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 1852–1859.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *CoRR* abs/1106.0675.

Vidal, V., and Geffner, H. 2004. Branching and pruning: An optimal temporal POCL planner based on constraint programming. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, 570–577.

Vidal, V., and Geffner, H. 2005. Solving simple planning problems with more inference and no search. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, 682–696.

Session 4

Planning and Scheduling

Task Scheduling and Trajectory Generation of Multiple Intelligent Vehicles

Jennifer David
Intelligent Systems Lab
School of Information Technology
Halmstad University
Halmstad, Sweden 30118
jendav@hh.se

Abstract

The problem of multi-vehicle path planning can be treated as a machine scheduling or a vehicle routing problem with definite temporal, spatial and other constraints. We consider the scenario of path planning of multiple Automated Guided Vehicles (AGVs) and Automated Trucks (ATs) in a Ship-Container Terminal area*. Our main focus is on developing global and local path planning methods for each of these vehicles along with a centralized task scheduler to designate goals to each of the vehicle. The whole problem can be divided in to two folds: Firstly, it is required to produce optimal schedule (planned path lengths), for a fleet of AGVs to certain goal points to pickup/drop off containers. It is treated similar to a truck scheduling problem with special constraints to deal with dynamic and complex environment. Secondly, it involves finding collision free trajectories for each of the vehicle using gradient descent method. We aim to propose an integrated framework for solving the goal assignment and trajectory planning problem minimizing the maximum cost over all vehicle trajectories and avoiding conflicts.

Research Situation

I started my PhD program in May 2014 at the Intelligent Systems Lab under the Center for Applied Intelligent Systems Research at Halmstad University. My supervisors are Prof. Thorsteinn Rognvaldsson and Dr. Rafael Valencia. I am also being by Dr. Karl Iagnemma (Visiting Professor from MIT, Cambridge, MA). Dr. Roland Philippsen of Google Inc., USA is my research mentor. My project is funded by EU Project Cargo-ANTs FP7-605598 in collaboration with three industrial partners Volvo AB - Sweden, TNO - Netherlands and ICT Automatisering Netherlands and academic partners IRI-CSIC, Spain. The project started on September 2013 and the aim of the project is to create smart Automated Guided Vehicles (AGVs) and Automated Trucks (ATs) that can co-operate in shared workspaces for efficient and safe freight transportation in main ports and freight terminals. The project ends on August 2016. The PhD program is generally 3-4 years and I two more years to defend my thesis. There is no research proposal stage and I have been working on the project from the start of my program.

My focus of my part of the project is on multi-vehicle path planning. The objective of the work is to develop and demonstrate planning, decision, control and safety strategies for automated vehicles. My main research question is to setup a high level interaction planning which integrates with the vehicle control system. So far, a complete navigation framework has been developed in this work as shown in figure 1. It consists of a task scheduler, global path planner and a local planner. Literature review has been conducted on each of these areas to choose on the specific approach/methodology that will fit our application scenario. This will be discussed in the third section. A simple task assignment algorithm based on the classical Hungarian Assignment (Kuhn 1955), a dynamic navigation global planner based on Estar (Philippsen 2002) and a local planner based on CHOMP (Covariant Hamiltonian Optimization Algorithm for Motion Planning) by Zucker 2012 has been adapted for this framework.

The novelty of my thesis is to adapt this approach for the chosen scenario which will be discussed in the next section. With respect to this, a simple ROS navigation stack has been developed in C++. Preliminary results of the local planner have been conducted on an Automated Truck Volvo FH 60 since November 2015 and have been continuously debugged. The ROS stack has been continuously extended to adapt to complex and dynamic environments. Presently, I have been integrating mapping and vehicle control techniques with this framework by conducting real time tests as well as simulation tests in Gazebo. High level planning and task scheduler are needed to be integrated yet. Moreover, research questions on task scheduler and novelty problems faced during integration have to be addressed yet.

Content and Motivation

More than 60% of the worlds cargo has been transported using ships and there is a continuous growth in global container trade. Hence, container transport industry faces new challenges such as increasingly stringent environmental regulations as well as capacity bottlenecks at ports and hinterland connections. The Cargo-ANTs project is based on

the premise that technological innovations in transshipment technologies will play a key role in meeting these demands in an integrative and cost efficient manner especially on the truck scheduling problem in container terminals.

In this regard, there is a need of a completely autonomous system to transport containers using AGVs and ATs. Hence, in this work, apart from developing global and local planner for making an AGV/AT fully autonomous, there is a need for creating an optimal task scheduler that resembles a traditional truck scheduler used in ports. However, this scheduler is integrated with the vehicle planners in such a way that it can avoid path conflicts with other vehicles as well as produces optimal paths in terms of the path length. This is the novelty of my thesis.

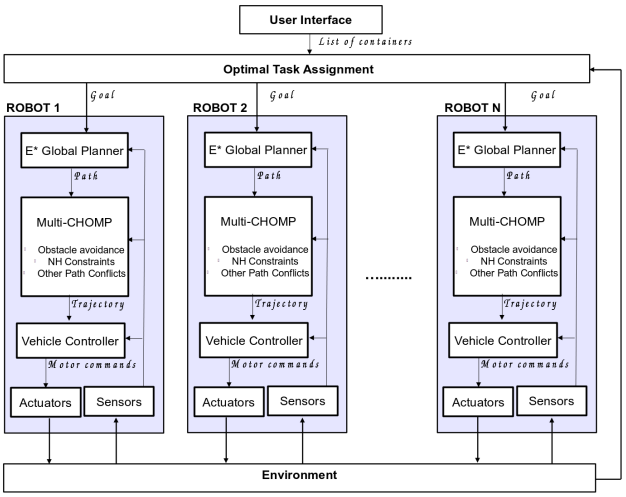


Figure 1: Navigational Framework

Related Work

The core part of the thesis is about Multi-Vehicle Path Planning. It can be dealt as a centralized planner or a decentralized planner. But, there is a tradeoff between complexity and optimality in these approaches. There are also other approaches as in swarm robotics as multi robot coordination problem but they are far from optimality. Recently, a new approach involves treating multi robot vehicle problem as task allocation problem which is a balance between centralized and decentralized approach that gives near optimal solutions. Liu and Shell (2013) used an any time assignment algorithm which was a dual to the Hungarian algorithm. Turpin et al. (2014) used an optimal goal assignment method coupled with trajectory planning for multiple quadcopters.

All of these approaches used only assignment methods especially the Hungarian or any variant of this algorithm. There is no scheduling involved that includes temporal and spatial constraints. Gombolay et al. (2009) developed a fast task sequencer in conjunction with a MILP solver to generate near-optimal task schedules. But there is no integration with trajectory planner here. Hence, this work tries to exploit the usage of a centralized task scheduler to avoid path conflicts between vehicles. Since task scheduler itself is a time complexity problem, coupling a planner with it, is a challenging problem to look on.

Problem Statement

The main research problem that will be addressed here is on exploiting the global nature of the task scheduler in avoiding local conflicts between vehicle paths by coupling it with the trajectory planner. This problem has not been discussed in the literature due to the computational complexity of the scheduler. However, if scheduler is coupled with trajectory planner, it is possible to develop near optimal solutions for multi-vehicle planning problems with less complexity.

Research Goals/Methodology

The whole of this work is a synergetic amalgamation of three different areas - optimization techniques in container terminal schedulers, multi robot task allocation and trajectory planners. Depending upon our application, a navigation framework is to be designed that could be run on real AGVs and ATs in a dynamic environment as the container terminal area. The research problem is to develop a centralized task scheduler similar to a traditional truck scheduler for container terminals with additional constraints suitable for autonomous vehicles. The second part of the research problem deals with the trajectory planner of the vehicle in optimizing the planned trajectory to adapt to the kinematic and non-holonomic constraints of the vehicle using a gradient descent method. The main research goals are:

- Development of a centralized task scheduler for autonomous vehicles.
- Development of a smooth and collision free trajectory planner.
- Integration of the scheduler with the trajectory planner to avoid conflicts.

We presently use Hungarian, variants of Hungarian and auction algorithm for task assignments in a simple environment. Hence, for task scheduler, we adapt an existing fast task sequencer (Gombolay et al. 2009) that can work for truck scheduling constraints. For obtaining smooth and collision free trajectories, a gradient descent method (CHOMP) is used. Adaptation of these algorithms for this specific application is one of the major research goals.

Dissertation Status

Literature survey has been done on three different areas - truck schedulers in container terminal areas, trajectory planners and multi-robot task allocation methods. With respect to the dissertation, adding side-slip constraints to the wheels of the vehicles has been implemented on the trajectory planner. The problem formulation of the truck scheduler with spatial and temporal constraints have been devised. Implementation of this scheduler instead of the regular task sequencer is under study. Literature study is being done on ways to couple scheduler with trajectory planner.

Expected Contributions

The expected outcome of this thesis is to find an approach that is computationally less expensive and produces near optimal solutions for multi vehicle path planning problems. Though the concept of task allocation has yielded better results in literature, we aim to replace it with task scheduler. By this, the application of this approach can be extended to more complex and dynamic environment with temporal constraints. Moreover, the global nature of the task scheduler can be used to resolve the local path conflicts between vehicles thus avoiding priority setting of vehicles.

References

Zucker, M., Ratliff, N., Dragan, A.D., Pivtoraiko, M., Klingensmith, M., Dellin, C.M., Bagnell, J.A. and Srinivasa, S.S., 2013. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10), pp.1164-1193.

Turpin, M., Mohta, K., Michael, N. and Kumar, V., 2014. Goal assignment and trajectory planning for large teams of interchangeable robots. *Autonomous Robots*, 37(4), pp.401-415.

Liu, L. and Shell, D.A., 2013. An anytime assignment algorithm: From local task swapping to global optimality. *Autonomous Robots*, 35(4), pp.271-286.

Kuhn, H.W., 1955. The Hungarian method for the assignment problem. Naval research logistics quarterly, 2(12), pp.83-97.

Philippsen, R. and Siegwart, R., 2005, April. An interpolated dynamic navigation function. In Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on (pp. 3782-3789). IEEE.

Gombolay, M.C., Wilcox, R. and Shah, J.A., 2013, June. Fast Scheduling of Multi-Robot Teams with Temporospatial Constraints. In Robotics: Science and Systems.

Acknowledgments

This work has been supported by the EU Project Cargo-Ants FP7-605598.

Decoupled State Space Search – Dissertation Abstract

Daniel Gnad
Saarland University
Saarbrücken, Germany
gnad@cs.uni-saarland.de

Abstract

Decoupled State Space Search is a recent approach to exploiting problem structure in classical planning. The particular structure needed is a star topology, with a single *center* component interacting with multiple *leaf* components. All interaction of the leaves with the rest of the problem has to be via the center. Given this kind of problem decomposition, we have showed that search on this *reformulated state space* can be exponentially more efficient than standard search. However, there do also exist cases in which decoupled search has to spend exponentially *more* effort to solve a task. We want to tackle this issue by combining decoupled search with different known search enhancement techniques, such as partial-order reduction, symmetry reduction, or dominance pruning. Presumably, these can be nicely combined with our new approach, such that we can prevent the exponential blow-up. Decoupled search is not restricted to classical planning, though. Its principles apply to all kinds of (heuristic) search problems, like, e. g., in Model Checking.

Introduction

In classical planning, heuristic search is a popular approach to solve a variety of input problems. The solution of such a planning task takes the form of a path, i. e., a sequence of transitions that lead from a given start state to a state satisfying certain goal conditions. To find such a path, (possibly huge) deterministic transition system, the task's *state space*, need to be explored. Inherent in this way of finding solutions to planning problems is the issue of state space explosion. We propose decoupled state space search (Gnad and Hoffmann 2015; Gnad *et al.* 2015) to solve this problem. Decoupled search can be seen as a form of *factored planning* (e. g., (Amir and Engelhardt 2003; Brafman and Domshlak 2006; 2008; 2013; Fabre *et al.* 2010)) , that restricts the interaction between the factors to take the form of a *star*, with a single center factor that interacts with multiple leaf factors. All interaction of a leaf with another factor has to be via the center. Hereby, decoupled search exploits a kind of conditional independence between the leaves – given a fixed center path π^C, *compliant* leaf paths can be scheduled independently along π^C. This allows for an efficient search and solution reconstruction, since complex cross-factor dependencies do not have to be resolved, which can make other factored planning approaches infeasible in practice. In our experiments, we observed that the decoupled state space, oftentimes is exponentially smaller than the standard state space.

There is also bad news, however. Although in most of our experiments we see an exponential reduction of the state space size under decoupled search, the state space can also get exponentially larger. This is so because of the special structure of the leaves, that "remember" the center path leading to the current decoupled state. Thus, when reaching the same state via different paths, decoupled search treats all these states as if they were different, leading to the blow-up. One possible means to circumvent this issue are dominance pruning methods. The simple method employed by (Gnad and Hoffmann 2015) already suffices to guarantee that the decoupled state space – which in principle can grow to infinite size – stays finite. Future work is going to derive more elaborate pruning methods that are more effective in reducing the size of the decoupled state space; the eventual goal being to upper bound its size by that of the standard state space.

Besides, many search enhancement techniques that have been proposed for standard state space search can probably also be deployed in the decoupled setting. Prominent topics in standard search are for example partial-order reduction, or symmetry, and dominance pruning.

Further more, although our theoretical framework allows for general star-shape factorings, in practice we only use fork and inverted-fork like structures. One of the reasons why we stuck to this is the complexity of computing general star factorings. Even if the only objective is the maximization of the number of leaf factors, obtaining such a factoring is **NP**-hard.

The rest of this work is organized as follows. The next chapter gives a brief summary of the relevant definitions of decoupled search as provided by (Gnad and Hoffmann 2015; Gnad *et al.* 2015). The reader familiar with decoupled search is invited to skip this chapter and proceed to *Future Work*, which introduces several lines of future research. We conclude with a brief summary.

Decoupled Search

Background

Our prior work has introduced Decoupled Search using a finite-domain state variable formalization of planning (e. g.

(Bäckström and Nebel 1995; Helmert 2006)). A *finite-domain representation* planning task, short FDR task, is a quadruple $\Pi = \langle V, A, I, G \rangle$. V is a set of *state variables*, where each $v \in V$ is associated with a finite domain $\mathcal{D}(v)$. We identify (partial) variable assignments with sets of variable/value pairs. A complete assignment to V is a *state*. I is the *initial state*, and the *goal* G is a partial assignment to V. A is a finite set of *actions*. Each action $a \in A$ is a triple $\langle \mathsf{pre}(a), \mathsf{eff}(a), \mathsf{cost}(a) \rangle$ where the *precondition* $\mathsf{pre}(a)$ and *effect* $\mathsf{eff}(a)$ are partial assignments to V, and $\mathsf{cost}(a) \in \mathbb{R}^{0+}$ is the action's non-negative *cost*.

For a partial assignment p, $\mathcal{V}(p) \subseteq V$ denotes the subset of state variables instantiated by p. For any $V' \subseteq \mathcal{V}(p)$, by $p[V']$ we denote the assignment to V' made by p. An action a is *applicable* in a state s if $\mathsf{pre}(a) \subseteq s$, i. e., if $s[v] = \mathsf{pre}(a)[v]$ for all $v \in \mathcal{V}(\mathsf{pre}(a))$. Applying a in s changes the value of each $v \in \mathcal{V}(\mathsf{eff}(a))$ to $\mathsf{eff}(a)[v]$, and leaves s unchanged elsewhere; the outcome state is denoted $s[\![a]\!]$. We also use this notation for partial states p: by $p[\![a]\!]$ we denote the assignment over-writing p with $\mathsf{eff}(a)$ where both p and $\mathsf{eff}(a)$ are defined. The outcome state of applying a sequence of (respectively applicable) actions is denoted $s[\![\langle a_1, \ldots, a_n \rangle]\!]$. A *plan* for Π is an action sequence s.t. $G \subseteq I[\![\langle a_1, \ldots, a_n \rangle]\!]$. The plan is *optimal* if its summed-up cost is minimal among all plans for Π.

To define factorings, we need the notion of the *causal graph* (e. g. (Knoblock 1994; Jonsson and Bäckström 1995; Brafman and Domshlak 2003; Helmert 2006)) using the commonly employed definition in the FDR context, where the causal graph *CG* is a directed graph over vertices V, with an arc from v to v', which we denote $(v \to v')$, if $v \neq v'$ and there exists an action $a \in A$ such that $(v, v') \in [\mathcal{V}(\mathsf{eff}(a)) \cup \mathcal{V}(\mathsf{pre}(a))] \times \mathcal{V}(\mathsf{eff}(a))$. In words, the causal graph captures precondition-effect as well as effect-effect dependencies, as result from the action descriptions. A simple intuition is that, whenever $(v \to v')$ is an arc in *CG*, changing the value of v' may involve changing that of v as well. We assume for simplicity that *CG* is weakly connected (this is wlog: else, the task can be equivalently split into several independent tasks).

We will also need the notion of a *support graph*, *SuppG*, similarly as used e. g. by (Hoffmann 2011). *SuppG* is like *CG* except its arcs are only those $(v \to v')$ where there exists an action $a \in A$ such that $(v, v') \in \mathcal{V}(\mathsf{pre}(a)) \times \mathcal{V}(\mathsf{eff}(a))$. In words, the support graph captures only the precondition-effect dependencies, not effect-effect dependencies. This more restricted concept will be needed to conveniently describe our notion of star topologies, for which purpose the effect-effect arcs in *CG* are not suitable.

Given the required notation, we can now define a factoring as a partition of the variables V into non-empty subsets $\mathcal{F}$, called factors.

Definition 1 (Star Factoring) *Let Π be an FDR task, and let $\mathcal{F}$ be a factoring. The* support-interaction graph *SuppIG($\mathcal{F}$) of $\mathcal{F}$ is the directed graph whose vertices are the factors, with an arc $(F \to F')$ if $F \neq F'$ and there exist $v \in F$ and $v' \in F'$ such that $(v \to v')$ is an arc in SuppG. $\mathcal{F}$ is a* star factoring *if $|\mathcal{F}| > 1$ and there exists $F^C \in \mathcal{F}$ s.t. the following two conditions hold:*

(1) The arcs in SuppIG($\mathcal{F}$) are contained in $\{(F^C \to F^L), (F^L \to F^C) \mid F^L \in \mathcal{F} \setminus \{F^C\}\}$.

(2) For every action a, if there exist $F^L_1, F^L_2 \in \mathcal{F} \setminus \{F^C\}$ such that $F^L_1 \neq F^L_2$ and $\mathcal{V}(\mathsf{eff}(a)) \cap F^L_1 \neq \emptyset$ as well as $\mathcal{V}(\mathsf{eff}(a)) \cap F^L_2 \neq \emptyset$, then $\mathcal{V}(\mathsf{eff}(a)) \cap F^C \neq \emptyset$.

F^C is the center *of $\mathcal{F}$, and all other factors $F^L \in \mathcal{F}^L := \mathcal{F} \setminus \{F^C\}$ are* leaves*. A star factoring $\mathcal{F}$ is* strict *if the arcs in IG($\mathcal{F}$) are contained in $\{(F^C \to F^L), (F^L \to F^C) \mid F^L \in \mathcal{F} \setminus \{F^C\}\}$.*

Note that *every* FDR task has a star factoring. In fact, any partition of the variables into two non-empty subsets is a star factoring: Calling one half of the variables the "center", and the other the "leaf", we have a (strict) star factoring, as Definition 1 does not apply any restrictions if there is a single leaf only. That said, it is not clear whether single-leaf factorings are useful in practice.

Example 1 *As an illustrative example, consider a transportation task with one package p, and two trucks t_A, t_B moving along three locations l_1, l_2, l_3 arranged in a line. The FDR planning task $\Pi = \langle V, A, I, G \rangle$ is defined as follows. $V = \{p, t_A, t_B\}$ where $\mathcal{D}(p) = \{A, B, l_1, l_2, l_3\}$ and $\mathcal{D}(t_A) = \mathcal{D}(t_B) = \{l_1, l_2, l_3\}$. The initial state is $I = \{p = l_1, t_A = l_1, t_B = l_3\}$, i. e., p and t_A start at l_1, and t_B starts at l_3. The goal is $G = \{p = l_3\}$. The actions (all with cost 1) are truck moves and load/unload:*

- *move(x, y, z) with precondition $\{t_x = y, p = x\}$ and effect $\{t_x = z\}$, where $x \in \{A, B\}$ and $\{y, z\} \in \{\{l_1, l_2\}, \{l_2, l_3\}\}$.*
- *load(x, y) with precondition $\{t_x = y, p = y\}$ and effect $\{p = x\}$, where $x \in \{A, B\}$ and $y \in \{l_1, l_2, l_3\}$.*
- *unload(x, y) with precondition $\{t_x = y, p = x\}$ and effect $\{p = y\}$, where $x \in \{A, B\}$ and $y \in \{l_1, l_2, l_3\}$.*

Observe that a truck can only move if the package is currently inside it. The causal graph is shown in Figure 1.

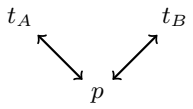


Figure 1: The causal graph of the example.

In this task, several factorings are possible. Consider, e. g., $\mathcal{F}_1 = \{\{t_A, t_B\}, \{p\}\}$, $\mathcal{F}_2 = \{\{t_A\}, \{t_B\}, \{p\}\}$, or $\mathcal{F}_3 = \{\{t_A\}, \{t_B, p\}\}$, all of which are clearly star factorings (though for F_2 only if we set $F^C = \{p\}$).

We are now defining the state space of a decoupled planning task. In contrast to standard search, decoupled search only branches over the center actions, enumerating what each leaf factor can do, separately. The *center actions* A^C are all those actions affecting the center. The *leaf actions* $A^L|_{F^L}$ for $F^L \in \mathcal{F}^L$ are all those actions affecting F^L. Observe that A^C *and* $A^L|_{F^L}$ *are not disjoint*, as the same action

may affect both A^C and $A^L|_{F^L}$. A leaf path is a sequence of leaf actions applicable to I when ignoring all center preconditions. A center path is a sequence of center actions applicable to I when ignoring all leaf preconditions.

After applying a center action to a state s, we update what is called the *pricing function* of s, $\mathsf{prices}[s]$. $\mathsf{prices}[s]$ assigns each leaf state of a leaf factor a price, representing the summed-up leaf action cost one would have to spend to reach the state from the initial state of the leaf factor. The update is performed for every reachable leaf state in s. We apply all leaf actions applicable given the center preconditions of $s[\![a]\!]$, updating $\mathsf{prices}[s[\![a]\!]]$ accordingly. More formally, the decoupled state space is defined as follows:

Definition 2 (Decoupled State Space) *Let Π be an FDR task, and $\mathcal{F}$ a star factoring with center F^C and leaves $\mathcal{F}^L$. A* decoupled state *s is a triple $\langle \mathsf{center}[s], \pi^C[s], \mathsf{prices}[s] \rangle$ where $\mathsf{center}[s]$ is a center state, $\pi^C[s]$ is a center path ending in $\mathsf{center}[s]$, and $\mathsf{prices}[s]$ is a* pricing function*, $\mathsf{prices}[s] : S^L \mapsto \mathbb{R}^{0+} \cup \{\infty\}$, mapping each leaf state to a non-negative* price*. The* decoupled state space *is a labeled transition system $\Theta^{\mathcal{F}}_{\Pi} = \langle S^{\mathcal{F}}, A^C, T^{\mathcal{F}}, I^{\mathcal{F}}, S^{\mathcal{F}}_G \rangle$ as follows:*

(i) $S^{\mathcal{F}}$ *is the set of all decoupled states.*

(ii) A^C*, the set of center actions, gives the transition labels.*

(iii) $T^{\mathcal{F}}$ *is the set of transitions, with $(s \xrightarrow{a^C} t) \in T^{\mathcal{F}}$ if: $a^C \in A^C$; $\pi^C[s] \circ \langle a^C \rangle = \pi^C[t]$; $\mathsf{pre}(a^C)[F^C] \subseteq \mathsf{center}[s]$ and $\mathsf{center}[s][\![a^C]\!] = \mathsf{center}[t]$; for every $F^L \in \mathcal{F}^L$ where $\mathsf{pre}(a^C)[F^L] \neq \emptyset$, there exists $s^L \in S^L|_{F^L}$ s.t. $\mathsf{pre}(a^C)[F^L] \subseteq s^L$ and $\mathsf{prices}[s](s^L) < \infty$; and, for every leaf $F^L \in \mathcal{F}^L$ and leaf state $s^L \in S^L|_{F^L}$, $\mathsf{prices}[t](s^L)$ is the cost of a cheapest path from $I[F^L]_0$ to s^L_n in $\mathsf{CompG}_{\Pi}[\pi^C[t], F^L]$, where $n := |\pi^C[t]|$.*

(iv) $I^{\mathcal{F}}$ *is the* decoupled initial state*, where $\mathsf{center}[I^{\mathcal{F}}] := I[F^C]$, $\pi^C[I^{\mathcal{F}}] := \langle\rangle$, and, for every leaf $F^L \in \mathcal{F}^L$ and leaf state $s^L \in S^L|_{F^L}$, $\mathsf{prices}[I^{\mathcal{F}}](s^L)$ is the cost of a cheapest path from $I[F^L]_0$ to s^L_0 in $\mathsf{CompG}_{\Pi}[\langle\rangle, F^L]$.*

(v) $S^{\mathcal{F}}_G$ *are the* decoupled goal states *s_G, where $\mathsf{center}[s_G]$ is a center goal state and, for every $F^L \in \mathcal{F}^L$, there exists a leaf goal state $s^L \in S^L|_{F^L}$ s.t. $\mathsf{prices}[s_G](s^L) < \infty$.*

We refer to paths $\pi^{\mathcal{F}}$ in $\Theta^{\mathcal{F}}_{\Pi}$ as decoupled paths*. A* solution *for $s \in S^{\mathcal{F}}$ is a decoupled path (denoted* GlobalPlan*) from s to some $s_G \in S^{\mathcal{F}}_G$. A solution for $\Theta^{\mathcal{F}}_{\Pi}$ is a called a solution for $I^{\mathcal{F}}$. A decoupled state, respectively $\Theta^{\mathcal{F}}_{\Pi}$, is* solvable *if it has a solution.*

Example 2 *In our example, when using the factoring $\mathcal{F} = \{\{p\}, \{t_a, t_B\}\}$ with center factor $F^C = \{t_A, t_B\}$, the initial state of the decoupled state space has only a single successor, resulting from $a^C = move(A, l_1, l_2)$. The precondition $p = A$ of $move(A, l_1, l_2)$ is reachable from $I[F^L]_0$ given the empty center path, but that is not so for the precondition $p = B$ of $move(B, l_3, l_2)$. This reflects the fact that, in the initial state of the task, we can move* only *the package (not moving a truck i. e. the center) so that $move(A, l_1, l_2)$ becomes applicable, but we cannot make $move(B, l_3, l_2)$ applicable in this manner.*

We don't provide the details of *compliant paths* and *compliant path graphs*, here, since this is out of scope for the presented work. Instead, we briefly summarize the intuition behind the notion of compliance.

A decoupled state s is a center path $\pi^C(s)$ associated for every leaf $F^L \in \mathcal{F}^L$ with the *π^C-compliant path graph* $\mathsf{CompG}_{\Pi}[\pi^C(s), F^L]$. Given $\pi^C(s)$ and a leaf path π^L, for π^L to be *compliant* with $\pi^C(s)$ we require that (1) the subsequences of shared actions in π^L and π^C coincide, and (2) in between, we can schedule π^L at monotonically increasing points alongside π^C s.t. (2a) the center precondition of each leaf action holds in the respective center state and (2b) the F^L precondition of each center action holds in the respective leaf state. The compliant path graph of s for a leaf F^L keeps track of *all* leaf paths in the leaf state space of F^L compliant with $\pi^C(s)$. In practice, we do not store the compliant path graphs for each state, but only the corresponding pricing functions.

For illustration, consider Example 3. The arcs from one layer to the next state that the surviving leaf states are only those which comply with $\mathrm{move}(A, l_1, l_2)$'s precondition, and will be mapped to possibly different leaf states by $\mathrm{move}(A, l_1, l_2)$'s effect. Within each layer the arcs correspond to those leaf-only actions whose center precondition is enabled at t. Note that, if $\mathrm{move}(A, l_1, l_2)$ had no precondition on F^L, then all leaf states would survive, and since $\mathrm{move}(A, l_1, l_2)$ has no effect on F^L, all leaf states remain the same at $t + 1$.

Example 3 *Consider again our illustrative example, using the factoring $\mathcal{F} = \{\{p\}, \{t_a, t_B\}\}$ with center factor $F^C = \{t_A, t_B\}$ and the center path $\pi^C = \langle move(A, l_1, l_2) \rangle$. The π^C-compliant path graph is shown in Figure 2.*

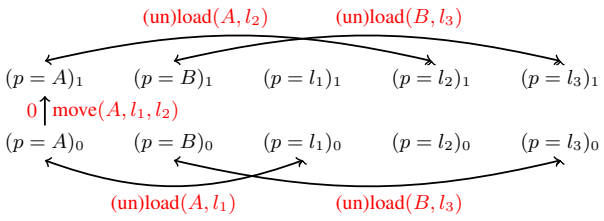


Figure 2: The compliant path graph for $\pi^C = \langle \mathrm{move}(A, l_1, l_2) \rangle$ in our illustrative example.

From layer 0 to layer 1, the only arc we have is that from $(p = A)_0$ to $(p = A)_1$. This is because $move(A, l_1, l_2)$ has precondition $p = A$, so all other values of p do not comply with the center action being applied at this layer, and are excluded from the compliant paths. Note that the arc has a weight of 0, because we do not account for the cost of center actions in the compliant path graph.

In our previous work, we showed that the plans for Π are in one-to-one correspondence with center paths augmented with compliant leaf paths. Say π is a plan for Π. The sub-sequence π^C of center actions in π is a center path. For a leaf $F^L \in \mathcal{F}^L$, the sub-sequence π^L of $A^L|_{F^L}$ actions in π is a leaf path. The sub-sequence of $A^C \cap A^L|_{F^L}$ actions in π^L coincides by construction with the sub-sequence of $A^C \cap A^L|_{F^L}$ actions in π^C. Furthermore, between any pair of subsequent shared actions, all F^C preconditions of π^L, and all F^L preconditions of π^C, must be satisfied because π is a plan, so we can read off an embedding, and π^L is π^C-compliant. Vice versa, say center path π^C ends in a goal center state, and can be augmented for every $F^L \in \mathcal{F}^L$ with a π^C-compliant leaf path π^L ending in a goal leaf state. Note that, if an action a affects more than one leaf, by the definition of star factorings a must also affect the center, so the sub-sequences of such actions are synchronized via π^C: They must be identical for every leaf involved, and correspond to the same action occurrences in π^C.

Overall, goal paths in the decoupled state space $\Theta^{\mathcal{F}}$ correspond to center goal paths augmented with compliant leaf goal paths, which correspond to plans for the original planning task Π, of the same cost. So (optimal) search in $\Theta^{\mathcal{F}}$ is a form of (optimal) planning for Π.

Heuristic Functions

In decoupled search, two different kinds of heuristic functions are of interest. *Center heuristics* h^C that estimate the remaining cost the center has to spend to reach the goal, and *star heuristics* $h^{\mathcal{S}}$ that estimate the overall remaining cost. We say that h is *center-admissible* if $h \leq h^{C*}$, and *star-admissible* if $h \leq h^{\mathcal{S}*}$. The conceptual distinction between h^C and $h^{\mathcal{S}}$ lies in that h^C cares only about how much work is left for the center factor, i. e., the cost of a center path sufficient to enable every leaf to reach its goal *somehow*. In contrast, $h^{\mathcal{S}}$ accounts for the combined cost of center and leaves, i. e. for the best extension of our current center path into an overall decoupled plan. We refer to heuristics attempting to estimate h^{C*} as *center heuristics*, and to heuristics attempting to estimate $h^{\mathcal{S}*}$ as *star heuristics*, and distinguish them notationally by superscripts "C" respectively "$\mathcal{S}$".

Observe that h^C is a special case of $h^{\mathcal{S}}$: We can compute h^C as $h^{\mathcal{S}}$ in a modified planning task where the cost of all leaf actions is set to 0. h^C keeps track only of which leaf states are reachable, not of the associated cost.

This may lead to qualitatively different decisions, i. e., h^C and $h^{\mathcal{S}}$ may *disagree*. Using a transportation example again, say that there are two alternative kinds of plans, (a) ones that pass the packages through several trucks, loading/unloading every time, vs. (b) ones that make more truck moves but have to load/unload each package only once and thus are better globally. Then h^C will draw search towards plans (a), whereas $h^{\mathcal{S}}$ will draw search towards plans (b).

Search Algorithms

Disregarding optimality, we can run any search algorithm on the decoupled state space, stopping at the first decoupled goal state. For optimal planning, matters are more subtle. One of our methods is formulated on a modified state space

Algorithm DX:
Input: FDR planning task Π, star factoring $\mathcal{F}$
Heuristic search algorithm X
star heuristic $h^{\mathcal{S}}$
Output: A plan for Π, or "failed"
Let $h^{\mathcal{S}}_{G\$} := \begin{cases} h^{\mathcal{S}}(s) & s \in S^{\mathcal{F}} \\ 0 & s = G' \end{cases}$
Run X with $h^{\mathcal{S}}_{G\$}$ on $\Theta^{\mathcal{F}}_{G\$}$
If X found a solution path $\pi = \pi^{\mathcal{F}} \circ \langle s_G \to G' \rangle$
return $\mathsf{GlobalPlan}(\pi^{\mathcal{F}})$
else return "failed"

Figure 3: Exploiting any known search algorithm X.

Algorithm *ADA**:
Input: FDR planning task Π, star factoring $\mathcal{F}$
Center heuristic h^C, star heuristic $h^{\mathcal{S}}$
Output: An optimal plan for Π, or "unsolvable"
Let $U := \infty$ /* best known upper bound */
Let $\pi^{\mathcal{F}}_U := \bot$ /* corresponding plan */
Run A* with h^C on $\Theta^{\mathcal{F}}$, with these modifications:
Continue search until the open list is empty
Whenever a goal vertex node $N[s_G]$ is expanded:
If $g(N) + \mathsf{Gprice}(s_G) < U$
let $U := g(N) + \mathsf{Gprice}(s_G)$
let $\pi^{\mathcal{F}}_U :=$ the decoupled plan leading to N
If $\mathsf{Gprice}(s_G) = \mathsf{MINGprice}$
return $\mathsf{GlobalPlan}(\pi^{\mathcal{F}}_U)$ /* early termination */
Whenever a node $N[s]$ is generated, and $U \neq \infty$:
If $g(N) + h^{\mathcal{S}}(s) \geq U$
discard N /* upper-bound pruning */
If $\pi^{\mathcal{F}}_U \neq \bot$ return $\mathsf{GlobalPlan}(\pi^{\mathcal{F}}_U)$ else return "unsolvable"

Figure 4: Anytime search algorithm. Search nodes are notated $N[s]$ where s is the state and N the node itself. MINGprice is the sum, over the leaf factors $F^L \in \mathcal{F}^L$, of optimal plan cost for the *projection* of Π onto F^L.

where the goal pricing functions are explicit. Consider Figure 3. DX ("*Decoupled* X") just runs any search algorithm X on $\Theta^{\mathcal{F}}_{G\$}$, which is defined as $\Theta^{\mathcal{F}}$ with the only difference that from every decoupled goal state $s_G \in S^{\mathcal{F}}_G$, we introduce a new transition to an artificial goal state G'. These transitions have cost $\mathsf{Gprice}(s_G) = \sum_{F_L \in \mathcal{F}} min\ \mathsf{prices}[s_G](s^L)$ with $s_L \in S^L|_{F^L}$ being a leaf goal state. If X is complete, then DX is complete. If X is optimal for admissible heuristics, then DX is optimal for star-admissible heuristics.

Consider now Figure 4. ADA* guides A* by a *center heuristic*, and uses a star heuristic merely for upper-bound pruning. The search is drawn to cheap center paths, disregarding leaf costs, so to guarantee optimality we must exhaust the open list. Without early termination, this would be dominated by DA* because ADA* would then have to expand at least all $N[s]$ where $g(N) + h^{\mathcal{S}}(s)$ is less than optimal solution cost. *With* early termination, that is not so because in the best case we have to exhaust only those

$N[s]$ where $g(N) + h^C(s)$ is less than optimal *center* solution cost. ADA* is complete, and is optimal for center-admissible h^C and star-admissible $h^{\mathcal{S}}$.

Future Work

Having introduced Decoupled Search in the previous chapter, we can now go into future work topics that arise more or less naturally from the pitfalls mentioned above. Besides, it is quite obvious that many search enhancement techniques that have been developed for standard search, can also be adapted to work in the decoupled setting.

A serious drawback of decoupled search, mentioned in the introduction, is that – in principle – the decoupled state space can be infinitely large. This is the case because the pricing functions implicitly remember the center path taken to the decoupled state. Consider a slightly changed version of our illustrative example, that does not require a package to be loaded in a truck to be able to $move$ the truck. With the factoring that has the package as the center and the two trucks as leaves, there are only 5 center states (the different positions of the package). In the initial state, i. e., with the empty center path, the trucks can be located at all three positions with a price of 0 to 2, depending on their initial position. Say we decide to load the package into t_B, using the compliant leaf state $t_B = l_1$ for a price of 2 and to immediately unload the package again, resulting in the decoupled state s_1. Then, the initial center state is identical to that of s_1, but their prices for t_B differ. For the initial state, they are $[2, 1, 0]$ for $[t_B = l_1, t_B = l_2, t_B = l_3]$ and for s_1 they are $[2, 3, 4]$. Consequently, we must consider s_1 to be a new decoupled state. However, it is intuitively obvious, that the initial state "is better" than s_1, more formally: Whatever we can do in s_1, we can at least as cheaply do in the initial state. The question arises what "better" means in this scenario, so how can we define a notion of *dominance* between two decoupled states s_1 and s_2? As already described, the simple notion of dominance employed by (Gnad and Hoffmann 2015) is sufficient to guarantee that the just characterized problem can no longer occur. The dominance check does a pointwise comparison of the prices of the two states and whenever the center states are identical and $\mathsf{prices}[s_1](s_L) \leq \mathsf{prices}[s_2](s_L)$ for all leaf states, s_1 dominates s_2 and we can discard s_2. However, this does not prevent the exponential blow-up of the decoupled state space compared to the standard one, yet. How to further reduce the size of the decoupled state space and possibly upper bound it by the size of the standard state space, is still an open question.

Closely related to this is the idea of not only being able to introduce dominance between decoupled states with the same center, but also when the center states differ. Recent work (Hall *et al.* 2013; Torralba and Hoffmann 2015) has introduced such kind of pruning in the standard state space. It will be a highly interesting research question how to adapt this to the decoupled setting.

In a similar direction goes the pruning of symmetric states, that has been used in planning for several years (e. g. (Pochter *et al.* 2011; Domshlak *et al.* 2012; 2013)). Can we detect symmetries between center states and perform pruning based on some criteria of their corresponding pricing functions? Or is it even possible to detect more global symmetries in the decoupled state space, e. g., considering a standard logistics example. If two packages are completely symmetric, there is no need to handle them separately, but they can be treated as a single package.

In some domains, our current factoring strategies do find factorings with a high number of leaves, but these are very small and the remaining center state space is not much smaller than the whole standard state space. In this case, it makes sense to think of methods further reducing the size of the center state space. One promising approach is partial-order reduction via strong stubborn sets, a technique originally proposed in the context of model checking, that has recently been introduced in planning (Valmari 1989; Wehrle and Helmert 2012; Wehrle *et al.* 2013; Wehrle and Helmert 2014). The inverted setting does occur, too. A factoring resulting in a small center factor where enumerating the state space of few very big leaves leads to a significant runtime overhead. In this case, does it make sense to apply partial-order reduction to the leaves?

Talking about the factoring strategies, so far we only deployed fork-like factorings, though our framework allows for much more general star topologies. Further exploring the space of possible factoring methods is an important task, making decoupled search applicable to a large number of new domains. This not only considers the application to other classical planning domains, but also extends the scope to other search-based areas such as puzzles, or multi-agent planning, but also model checking.

The combination of decoupled search with other known techniques from, e. g., model checking is highly interesting, too. How about a kind of hybrid search that performs standard search in the center part and symbolic search in the leaves? Especially if the leaves have a rather large state space, as, e. g., is very likely if we look at 2-factor star topologies, this promises to solve the problem of the big overhead needed to compute and store the pricing functions.

Overall, there is a great variety of techniques and methods, combining which with decoupled search can enrich the state-of-the-art in planning and related search applications.

Conclusion

Decoupled state space search promises a wide open new research area, exploring which is far beyond the scope of a single dissertation. The initial framework has already been introduced and tightened, so new extensions have a stable basis to build on. The enhancement techniques outlined in the previous chapter are mostly known from standard state space search and can presumably be combined with decoupled search. Being orthogonal to the idea of decoupling – which is exploiting conditional independence between parts of a planning task – a significant additional improvement over the plain variant can be expected when enabling these methods in decoupled search.

References

Eyal Amir and Barbara Engelhardt. Factored planning. In G. Gottlob, editor, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 929–935, Acapulco, Mexico, August 2003. Morgan Kaufmann.

Christer Bäckström and Bernhard Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

Ronen Brafman and Carmel Domshlak. Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research*, 18:315–349, 2003.

R. I. Brafman and C. Domshlak. Factored planning: How, when, and when not. In Yolanda Gil and Raymond J. Mooney, editors, *Proceedings of the 21st National Conference of the American Association for Artificial Intelligence (AAAI-06)*, pages 809–814, Boston, Massachusetts, USA, July 2006. AAAI Press.

Ronen I. Brafman and Carmel Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, editors, *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, pages 28–35. AAAI Press, 2008.

Ronen Brafman and Carmel Domshlak. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence*, 198:52–71, 2013.

Carmel Domshlak, Michael Katz, and Alexander Shleyfman. Enhanced symmetry breaking in cost-optimal planning as forward search. In Blai Bonet, Lee McCluskey, José Reinaldo Silva, and Brian Williams, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press, 2012.

Carmel Domshlak, Michael Katz, and Alexander Shleyfman. Symmetry breaking: Satisficing planning and landmark heuristics. In Daniel Borrajo, Simone Fratini, Subbarao Kambhampati, and Angelo Oddi, editors, *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, Rome, Italy, 2013. AAAI Press.

Eric Fabre, Loïg Jezequel, Patrik Haslum, and Sylvie Thiébaux. Cost-optimal factored planning: Promises and pitfalls. In Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz, editors, *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, pages 65–72. AAAI Press, 2010.

Daniel Gnad and Jörg Hoffmann. Beating LM-cut with h^{max} (sometimes): Fork-decoupled state space search. In Ronen Brafman, Carmel Domshlak, Patrik Haslum, and Shlomo Zilberstein, editors, *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*. AAAI Press, 2015.

Daniel Gnad, Jörg Hoffmann, and Carmel Domshlak. From fork decoupling to star-topology decoupling. In Levi Lelis and Roni Stern, editors, *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*. AAAI Press, 2015.

David Hall, Alon Cohen, David Burkett, and Dan Klein. Faster optimal planning with partial-order pruning. In Daniel Borrajo, Simone Fratini, Subbarao Kambhampati, and Angelo Oddi, editors, *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, Rome, Italy, 2013. AAAI Press.

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

Jörg Hoffmann. Analyzing search topology without running any search: On the connection between causal graphs and h^+. *Journal of Artificial Intelligence Research*, 41:155–229, 2011.

Peter Jonsson and Christer Bäckström. Incremental planning. In *European Workshop on Planning*, 1995.

Craig Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.

Nir Pochter, Aviv Zohar, and Jeffrey S. Rosenschein. Exploiting problem symmetries in state-based planners. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI-11)*, San Francisco, CA, USA, July 2011. AAAI Press.

Álvaro Torralba and Jörg Hoffmann. Simulation-based admissible dominance pruning. In Qiang Yang, editor, *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 1689–1695. AAAI Press/IJCAI, 2015.

Antti Valmari. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets*, pages 491–515, 1989.

Martin Wehrle and Malte Helmert. About partial order reduction in planning and computer aided verification. In Blai Bonet, Lee McCluskey, José Reinaldo Silva, and Brian Williams, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press, 2012.

Martin Wehrle and Malte Helmert. Efficient stubborn sets: Generalized algorithms and selection strategies. In Steve Chien, Minh Do, Alan Fern, and Wheeler Ruml, editors, *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Press, 2014.

Martin Wehrle, Malte Helmert, Yusra Alkhazraji, and Robert Mattmüller. The relative pruning power of strong stubborn sets and expansion core. In Daniel Borrajo, Simone Fratini, Subbarao Kambhampati, and Angelo Oddi, editors, *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, Rome, Italy, 2013. AAAI Press.

Hierarchical Task Model with Alternatives for Predictive-reactive Scheduling

Marek Vlk and Roman Barták (supervisor)
Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
{vlk, bartak}@ktiml.mff.cuni.cz

Abstract

Attaining optimal results in real-life scheduling is hindered by a number of problems. One such problem is dynamics of scheduling environments with breaking-down resources and hot orders coming during the schedule execution. Traditional approach to react to unexpected events occurring on the shop floor is generating a new schedule from scratch. Complete rescheduling, however, may require excessive computation time. Moreover, the recovered schedule may deviate a lot from the ongoing schedule. Some work has focused on tackling these shortcomings, but none of the existing approaches tries to substitute jobs that cannot be executed with a set of alternative jobs. This paper describes the scheduling model suitable for dealing with unforeseen events using the possibility of alternative processes and states the future goals.

Introduction

Scheduling, the aim of which is to allocate scarce resources to activities in order to optimize certain objectives, has been frequently addressed in the past decades. Developing a detailed schedule in manufacturing environment helps maintain efficiency and control of operations.

In the real world, however, manufacturing systems face uncertainty owing to unforeseen events occurring on the shop floor. Machines break down, operations take longer than anticipated, personnel do not perform as expected, urgent orders arrive, others are canceled, etc. These disturbances may bring inconsistencies into the ongoing schedule. If the ongoing schedule becomes infeasible, the simple approach is to collect the data from the shop floor when the disruption occurs and to generate a new schedule from scratch. Because most of the scheduling problems are NP-hard, complete rescheduling usually involves prohibitive computation time and an excessive deviation of the recovered schedule from the original schedule.

To avoid the problems of rescheduling from scratch, reactive scheduling, which may be conceived as the continuous correction of precomputed predictive schedules, is becoming more and more important. Reactive scheduling is contradistinguished from predictive scheduling mainly by its on-line nature and associated real-time execution requirements. The schedule update must be accomplished before the running schedule becomes invalid, and this time window may be very short in complex scheduling environments.

Several novel sophisticated methods attempt to cope with the shortcomings of complete rescheduling, e.g., by rescheduling only the activities somehow affected by the disturbance. To the best of our knowledge, however, none of the existing approaches tries to replace some activities by a set of alternative activities (using other available resources) to achieve the same goal.

In this paper we propose a model suitable for development of algorithms for modifying a schedule to accommodate disturbances, such as a machine breakdown, using the possibility of alternative processes, i.e., to re-plan the influenced part of the schedule.

Related Work

The approaches how to tackle dynamics of the scheduling environment can be divided basically into two branches according to whether or not the predictive schedule is computed before the execution starts (Vieira, Herrmann, and Lin 2003). If the predictive schedule is not computed beforehand and individual activities are assigned to resources pursuant to some so called dispatching rules during the execution, we talk about *completely reactive scheduling* or on-line scheduling. This strategy is suitable for very dynamic environments, where it is not known in advance which activities it will be necessary to process. On the other hand, it is obvious that this approach hardly ever leads to an optimal or near-optimal schedule.

If the schedule is crafted beforehand and then updated during its execution, it is referred to as *predictive-reactive scheduling*. When correcting the ongoing schedule in response to changes within the environment, the aim is usually to minimize the schedule modification. The motivation for minimizing the alteration of the schedule is that every aberration may lead to deterioration in the performance owing to affecting other planning activities based upon the original schedule. Similarity of two schedules may be formally defined for example as a *minimal perturbation problem* (Barták, Muller, and Rudová 2003).

There is an extensive literature on rescheduling (Ouelhadj and Petrovic 2009; Raheja and Subramaniam 2002). First,

the *heuristic-based* approaches do not guarantee finding an optimal solution, but they respond in a short time. The simplest schedule-repair technique is the *right shift rescheduling* (Abumaizar and Svestka 1997), which shifts the operations globally to the right on the time axis in order to cope with disruptions. This may lead to schedules of very bad quality.

Another simple heuristic is *affected operation rescheduling* (Smith 1995), also referred to as partial schedule repair, the essence of which is to reschedule only the operations directly and indirectly affected by the disruption in order to minimize the deviation from the initial schedule.

Better schedules to the detriment of computational efficiency may be attained by using *meta-heuristics* such as simulated annealing, genetic algorithms, tabu search, and iterative flattening search (Oddi et al. 2007). These high level heuristics guide local search methods to escape from local optima by occasional accepting worse solutions or by generating better initial solutions for local probing in some sophisticated way.

Some techniques from the field of artificial intelligence and knowledge-based systems are also applied in rescheduling, namely *case-based reasoning* (Cunningham and Smyth 1997), *fuzzy logic* (Ramkumar, Tamilarasi, and Devi 2011), and *neural networks* (Jain and Meeran 1998). Another approach, which is rather an independent branch, is *multi-agent based architectures* (Zhang et al. 2011). Multi-agent systems seem to be the most promising approach, but the coordination among the agents is hard to achieve.

The attempts to absorb certain amount of uncertainty based on the past executions of schedules is considered in another strategy, usually referred to as *robust proactive scheduling*. One such example is a model and an algorithm generating a predictive schedule of production workflows that is (proactively) robust with regard to so called immediate events, which include breakdown of a workstation and faulty termination of a workflow execution (Dulai and Werner-Stark 2015). The robustness is attained by shifting activities (traditional introducing or enlarging gaps on resources) based on the probabilities of resource failures, which are estimated according to previous experiences. The drawback of the algorithm is the assumption that every resource failure is only temporary, and the time for how long the resource is unavailable in case of its failure is known in advance.

While there is a great amount of work devoted to planning with time and resources and to integrating planning and scheduling techniques, to the best of our knowledge, there is no research carried out aiming at the possibility of re-planning in the field of predictive-reactive scheduling.

Scheduling Model Description

The scheduling model we work with is taken from the FlowOpt system (Barták et al. 2012), which contains a tool for designing and editing *manufacturing workflows*. Workflow in general may be understood as a scheme of performing some complex process, itemized into simpler processes and relationships among them. Manufacturing workflow is then an outline how to obtain a desired product.

In order to make editing of workflows easier, the workflows in our model match up the structure of Nested Tem-

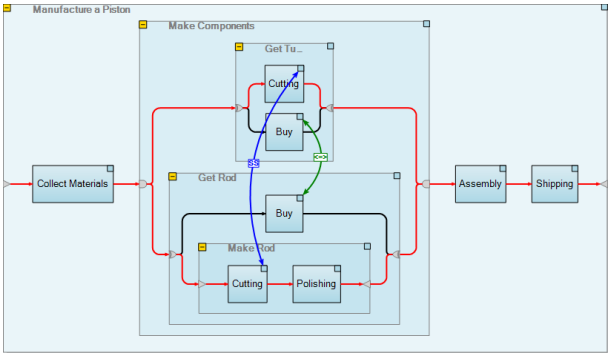


Figure 1: An example of a workflow (Skalický 2011).

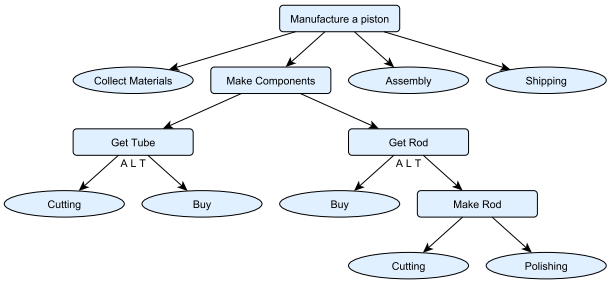


Figure 2: A decomposition tree for the workflow. The label "ALT" beneath tasks stands for alternative decomposition; the other decompositions are parallel. Activities are ellipse-shaped.

poral Networks with Alternatives (Barták and Čepek 2007), where the nodes of a network correspond to the tasks of a workflow. The tasks decompose into other tasks, referred to as their children. There are two types of decomposition: parallel and alternative. The tasks that do not decompose further (i.e., leaves) are called *primitive tasks*. The primitive tasks correspond to activities (or operations) and are associated with some additional parameters, namely start, end, and duration.

An example of a workflow and its decomposition tree are depicted in Figures 1 and 2. It contains eight primitive tasks (activities), three parallel tasks, and two alternative tasks.

The workflows as described define a number of feasible processes. A *process* is a subset of tasks selected to be processed. While a *parallel task* requires all its children to be processed, an *alternative task* requires exactly one of its children to be processed. If an arbitrary task is not in a process, none of its offspring is in the process either. Hence, to ensure that an instance of a workflow is actually processed, its root task has to be in the selected process. An example of a process is depicted in Figure 3.

To introduce some restrictions in terms of occurrences of tasks in the process and their time data, a pair of tasks can be bound by a *constraint*. Temporal constraints include *precedences* (one task has to be accomplished before the execution of another task starts), and *synchronizations* (one task has to

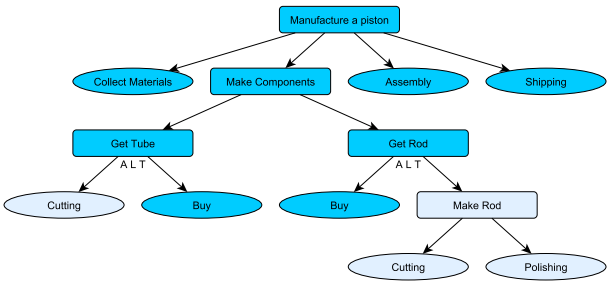


Figure 3: The decomposition tree with a selected process.

start/end exactly when another task starts/ends). Logical constraints include *implications* (if one task is in the process, the other task must be in the process too), *equivalences* (either both tasks must be in the process or none of them can be in the process), and *mutual exclusions* (at most one of the tasks can be in the process).

In Figure 1, there is one equivalence constraint enforcing that a tube and a rod are both either bought or made. The other constraint (synchronization) ensures that the activities "cutting tube" and "cutting rod" start at the same time.

Besides these constraints added by a user, which are referred to as custom constraints, there are some implicit constraints arising from the hierarchical structure of tasks. For example, the start time of a task equals the start time of its earliest child, and the end time of the task equals the end time of its latest child.

Activities are processed on *resources*. All resources are unary, which means that each resource may perform no more than one activity at a time. This limitation is often referred to as a resource constraint and belongs to the mentioned implicit constraints. Each activity is specified by a set of resources on which the activity can be processed (resource group), and in the resulting schedule, each activity in the selected process must be allocated to exactly one resource (selected resource).

Note that workflow is only a guideline how to manufacture a particular product. If a user wants n such products, n instances of the corresponding workflow are inserted into the model. An instance of a workflow is referred to as an *order*. It contains some additional data, such as due date and lateness penalty.

Finally, a resulting schedule is *feasible* if all custom as well as implicit constraints are satisfied.

Scheduling Problem

Formally, schedule S is a triplet of three sets: $Tasks$, $Constraints$, and $Resources$.

Tasks Tasks match up a forest structure. Therefore, every task T either has a parent, i.e., $\exists P \in Tasks : parent(T) = P$, or is a root, i.e., $parent(T) = null$.

- $subtasks(T) = \{C \in Tasks \mid parent(C) = T\}$
- $Roots = \{R \in Tasks \mid parent(R) = null\}$

There are three types of tasks: *parallel*, *alternative*, and *primitive* tasks.

- $Tasks = Parallel \cup Alternative \cup Primitive$
- $\forall T \in Parallel \cup Alternative : subtasks(T) \neq \emptyset$
- $\forall T \in Primitive : subtasks(T) = \emptyset$

Let process $P \subseteq Tasks$ be the set of tasks selected to be processed. Making the process feasible introduces the following constraints:

- $T \in P \cap Parallel : subtasks(T) \subseteq P$
- $T \in P \cap Alternative : |subtasks(T) \cap P| = 1$
- $T \notin P : subtasks(T) \cap P = \emptyset$

Let S_i and E_i denote the start time and end time, respectively, of task T_i. Each activity corresponding to the primitive task T_i is specified by the duration D_i. Then the time data are computed as follows:

- $\forall T_i \in P \cap Primitive : S_i + D_i = E_i$
- $\forall T_i \in P \cap (Parallel \cup Alternative) :$
  $S_i = \min\{S_j \mid T_j \in subtasks(T_i) \cap P\}$
  $E_i = \max\{E_j \mid T_j \in subtasks(T_i) \cap P\}$

Constraints There are two basic types of constraints: temporal, and logical. Temporal constraints restrict mutual position in time of two distinct activities. We take into consideration precedence and synchronization constraints, the semantics of which is as follows:

- $(i \rightarrow j) : T_i, T_j \in P \Rightarrow E_i \leq S_j$
- $(i\ ss\ j) : T_i, T_j \in P \Rightarrow S_i = S_j$
- $(i\ se\ j) : T_i, T_j \in P \Rightarrow S_i = E_j$
- $(i\ es\ j) : T_i, T_j \in P \Rightarrow E_i = S_j$
- $(i\ ee\ j) : T_i, T_j \in P \Rightarrow E_i = E_j$

Logical constraints are of three types: implications, equivalences, and mutexes. The semantics of the constraints is such:

- $(i \Rightarrow j) : T_i \in P \Rightarrow T_j \in P$
- $(i \Leftrightarrow j) : T_i \in P \Leftrightarrow T_j \in P$
- $(i\ mutex\ j) : T_i \notin P \vee T_j \notin P$

Resources Let $T \in Primitive$, then the set of resources that may process the primitive task T is denoted $Resources(T)$. The set $Resources(T)$ is often referred to as a resource group.

Each activity to be processed needs to be allocated to exactly one resource from its resource group. Let $T \in Primitive$, then a resource $R \in Resources(T)$ is *selected* if resource R is scheduled to process the primitive task T, which we denote $SelectedResource(T) = R$.

Each primitive task that is selected to the process P must have a selected resource to make a schedule feasible. Formally:

$$\forall T \in P \cap Primitive : SelectedResource(T) \neq null$$

All resources in a schedule are unary, which means that they cannot execute more tasks simultaneously. Therefore,

in a feasible schedule for all selected primitive tasks $T_i \neq T_j$ the following holds:

$$SelectedResource(T_i) = SelectedResource(T_j) \Rightarrow E_i \leq S_j \vee E_j \leq S_i$$

Schedule

A schedule S (sometimes referred to as a resulting schedule or a solution) is acquired by determining the set P, and allocating the primitive tasks from P in time and on resources. Allocation in time means assigning particular values to the variables S_i and E_i for each $T_i \in P$. Allocation on resources means selecting a particular resource ($SelectedResource(T)$) from the resource group ($Resources(T)$) of each task $T \in P \cap Primitive$.

To make a schedule *feasible*, the allocation must be conducted in such a way that all the mentioned constraints in the problem are satisfied.

Rescheduling Problem

The problem we actually deal with is that we are given a particular instance of the scheduling problem along with a feasible schedule, and also with a change in the problem specification. The aim is to find another schedule that is feasible in terms of the new problem definition. The feasible schedule we are given is referred to as an original schedule or an ongoing schedule.

Formally, let $R = (Pr_0, Sch_0, \delta^+, \delta^-)$ be a rescheduling problem, which is given by the original scheduling problem Pr_0, the original feasible schedule Sch_0, elements δ^+ to be added to the problem Pr_0, and elements δ^- to be removed from the problem Pr_0. New scheduling problem Pr_1 is then $Pr_0 \cup \delta^+ \setminus \delta^-$. The task of the rescheduling problem R is then to find a schedule Sch_1 for problem Pr_1, the quality of which is measured with respect to the original schedule Sch_0.

The way the scheduling problem can be modified depends on the disturbance. In case of a resource failure, we are given a resource that cannot be used (from a certain time point) while the set of orders remains unchanged, therefore $\delta^+ = \emptyset$ and δ^- is the broken down resource.

Another example is an urgent order arrival, which is a disturbance where an order (a set of workflow instances) is added into the model, and the aim is to update the ongoing schedule in such a way that the added order is accomplished as early as possible. In this case δ^+ is the new order (including constraints among new tasks) and $\delta^- = \emptyset$.

As explained in the introduction, regardless of what the optimization objective of the original schedule is, it seems to be wise to modify the schedule in such a way that the new schedule is as similar to the original one as possible. For this purpose we need to evaluate the modification distance.

Let P_z denote the selected process P in the schedule Sch_z, and S_i^z denote the start time of task T_i in schedule Sch_z. Then, apart from computation time, we take into account the following distance functions:

$$f_0 = |\{T \in P_1 \setminus P_0\}| + |\{T \in P_0 \setminus P_1\}|$$

$$f_1 = \sum_{T_i \in P_0 \cap P_1 \cap Primitive} |S_i^1 - S_i^0|$$

$$f_2 = |\{T_i \in P_0 \cap P_1 \cap Primitive \mid S_i^1 \neq S_i^0\}|$$

$$f_3 = \max_{T_i \in P_0 \cap P_1 \cap Primitive} |S_i^1 - S_i^0|$$

In words, f_0 is the number of different tasks in the schedules, f_1 measures the total sum of time shifts of activities, f_2 counts the number of shifted activities, and f_3 is the biggest time shift of an activity. There are other conceivable distance functions, but we concentrate on these ones.

Current Work

Our recent work (Barták and Vlk 2015) proposes two methods to handle a resource failure occurring on the shop floor during the schedule execution. The first method, *Right Shift Affected*, takes the activities that were to be processed on a broken machine, reallocates them, and then it keeps repairing violated constraints until it gets a feasible schedule. This approach is suitable when it is desired to move as few activities as possible, that is, minimizing the distance function f_2.

The second method, which is aimed at shifting activities by a short time distance regardless of the number of moved activities (that is, minimizing the distance functions f_1 and f_3), is called *STN-recovery*. The routine deallocates a subset of activities and then it allocates the activities again through integrating techniques from the field of constraint programming, namely *Conflict-Directed Backjumping with Backmarking* (Kondrak and Van Beek 1997). Before the allocation process, the search space is suitably pruned based on the values from the original schedule, which is another thing that seems to be neglected in the related literature.

The shortcoming of both the algorithms is that they neglect the possibility of alternative processes, which in practice may lead to a schedule recovery at a blow. Moreover, if the ongoing schedule is not recoverable, the algorithms are not able to securely report it and terminate.

Future Plans

We are currently developing algorithms under the hierarchical model described, where, in response to unexpected events, the intention is not only to modify the allocation of activities of the selected process, but to replace tasks in the process by other tasks that are not in the process, i.e., to re-plan some (ideally the smallest necessary, hence the motivation for the distance function f_0) subset of the schedule.

The first algorithm to try will work as follows. First, find the feasible process from all the orders in the schedule top-down, preferring the branches from the original schedule whenever possible. Second, after the process is selected, allocate activities from the process in time and on resources. If the second step fails, go back to the first step. Iterate until the schedule is found. For the second step, the crucial part of the STN-recovery algorithm mentioned above may be employed.

One natural improvement might be trying to allocate an activity straightaway when the corresponding primitive task is considered to be selected to the process. If the activity is successfully allocated, the searching for the process proceeds, otherwise it backtracks for alternatives.

However, our main effort will be made towards updating the schedule as locally as possible. If a resource suddenly becomes unavailable, it may suffice to merely replace the affected activities by alternatives that are just one level (one task decomposition) from the affected activities. It follows that the process should not be discarded and sought again top-down as described above, but it should be explored in a bottom-up fashion (from the primitive tasks upwards). The main difficulty of this approach are the logical constraints that must be propagated whenever the membership of a task in the process is being flipped.

In further future, the target is to extend the model of Nested Temporal Networks with Alternatives by recursion, and to suggest algorithms for this model. The recursion will bring the full power of planning, i.e., the possibility to generate tasks according to a given target. The main inspiration comes from the Hierarchical Task Networks (Nau et al. 2003). We will try modeling problems by attribute grammars, where modeling relations among attributes will be realized by constraint satisfaction problem rather than traditional semantic rules (Barták 2016).

Acknowledgments

This research is partially supported by SVV project number 260 224 and by the Czech Science Foundation under the project P103-15-19877S.

References

Abumaizar, R. J., and Svestka, J. A. 1997. Rescheduling job shops under random disruptions. *International Journal of Production Research* 35(7):2065–2082.

Barták, R., and Čepek, O. 2007. Nested temporal networks with alternatives. In *AAAI Workshop on Spatial and Temporal Reasoning, Technical Report WS-07-12, AAAI Press*, 1–8.

Barták, R., and Vlk, M. 2015. Machine breakdown recovery in production scheduling with simple temporal constraints. In *Agents and Artificial Intelligence*. Springer. 185–206.

Barták, R.; Jaška, M.; Novák, L.; Rovenský, V.; Skalický, T.; Cully, M.; Sheahan, C.; and Thanh-Tung, D. 2012. Flowopt: Bridging the gap between optimization technology and manufacturing planners. In *Luc De Raedt et al. (Eds.): Proceedings of 20th European Conference on Artificial Intelligence (ECAI 2012)*, 1003–1004. IOS Press.

Barták, R.; Muller, T.; and Rudová, H. 2003. Minimal perturbation problem-a formal view. *Neural Network World* 13(5):501–512.

Barták, R. 2016. Using attribute grammars to model nested workflows with extra constraints. In *SOFSEM 2016: Theory and Practice of Computer Science*. Springer. 171–182.

Cunningham, P., and Smyth, B. 1997. Case-based reasoning in scheduling: reusing solution components. *International Journal of Production Research* 35(11):2947–2962.

Dulai, T., and Werner-Stark, Á. 2015. A database-oriented workflow scheduler with historical data and resource substitution possibilities. In *Proceedings of the International Conference on Operations Research and Enterprise Systems*, 325–330. SciTePress.

Jain, A. S., and Meeran, S. 1998. Job-shop scheduling using neural networks. *International Journal of Production Research* 36(5):1249–1272.

Kondrak, G., and Van Beek, P. 1997. A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence* 89(1):365–387.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *J. Artif. Intell. Res.(JAIR)* 20:379–404.

Oddi, A.; Policella, N.; Cesta, A.; and Smith, S. F. 2007. Boosting the performance of iterative flattening search. In *AI* IA 2007: Artificial Intelligence and Human-Oriented Computing, LNCS 4733*. Springer Verlag. 447–458.

Ouelhadj, D., and Petrovic, S. 2009. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling* 12(4):417–431.

Raheja, A. S., and Subramaniam, V. 2002. Reactive recovery of job shop schedules – a review. *International Journal of Advanced Manufacturing Technology* 19:756–763.

Ramkumar, R.; Tamilarasi, A.; and Devi, T. 2011. Multi criteria job shop schedule using fuzzy logic control for multiple machines multiple jobs. *International Journal of Computer Theory and Engineering* 3(2):282–286.

Skalický, T. 2011. Interactive scheduling and visualisation. Master's thesis, Charles University in Prague.

Smith, S. F. 1995. Reactive scheduling systems. In *D. Brown and W. Scherer (eds.), Intelligent scheduling systems*, 155–192. Springer US.

Vieira, G.; Herrmann, J.; and Lin, E. 2003. Rescheduling manufacturing systems: a framework of strategies, policies, and methods. *Journal of Scheduling* 6:39–62.

Zhang, L.; Wong, T.; Zhang, S.; and Wan, S. 2011. A multi-agent system architecture for integrated process planning and scheduling with meta-heuristics. In *Proceedings of the 41st International Conference on Computers & Industrial Engineering*.

Dissertation Abstract: Numeric Planning

Johannes Aldinger
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Georges-Köhler-Allee 52
79110 Freiburg, Germany
aldinger@informatik.uni-freiburg.de

Extended Abstract[1]

Planning is the art to automatically find solutions to problems where a model of the world is described in terms of variables and actions. A solution to such a planning problem is a sequence of actions (called *plan*) transforming the initial situation to a state that satisfies a goal description. In *classical planning*, the world is described by Boolean variables and the field is well studied with eminent advances of the state of the art in the last decades. However, classical planning is not expressive enough, e.g. for many interesting real-world applications that rely on numeric quantities. Therefore, we contemplate on numeric planning in this dissertation.

Background

For planning, the domain description language PDDL is the standard to model planning problems. Fox and Long (2003) extended this language in order to model more expressive planning problems. In PDDL2.1, the expressiveness of the planning language is classified into layers. Classical planning problems can be expressed in layer 1. Layer 2 allows for numeric, rational valued, variables and can therefore express physical properties (such as the velocity of a vehicle) as well as resources (such as the fuel level of a vehicle). In layer 3, actions can have a duration in order to model planning problems requiring the concurrent execution of actions. Changes to the world happen at specific instants (start of action, end of action), an assumption that is lifted in layer 4, where continuous change of variables (e.g. the concurrent filling and draining of a tub) can be modeled as well. Finally, layer 5 allows for exogenous events: the world is dynamic and events can happen without the influence of the planning agent. This dissertation aims at shedding light on numeric planning expressible with PDDL2.1, layer 2, also known as numeric planning with instantaneous actions.

A numeric planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is a 4-tuple where $\mathcal{V}$ is a set of numeric variables v with domain $\mathbb{Q}^\infty := \mathbb{Q} \cup \{-\infty, \infty\}$. $\mathcal{O}$ is a set of operators, $\mathcal{I}$ the initial state and $\mathcal{G}$ the goal condition. A *numeric expression* $\mathsf{e}_1 \circ \mathsf{e}_2$ is an arithmetic expression with operators $\circ \in \{+, -, \times, \div\}$ and expressions e_1 and e_2 recursively defined over variables $\mathcal{V}$ and constants from $\mathbb{Q}$. A *numeric constraint* $(\mathsf{e}_1 \bowtie \mathsf{e}_2)$ compares numeric expressions e_1 and e_2 with $\bowtie \in \{\leq, <, =, \neq\}$. A *condition* is a conjunction of propositions and numeric constraints. A *numeric effect* is a triple $(\mathsf{v} \circ\!= \mathsf{e})$ where $\mathsf{v} \in \mathcal{V}$, $\circ\!= \in \{:=, +=, -=, \times=, \div=\}$ and e is a numeric expression. Operators $o \in \mathcal{O}$ are of the form $\langle \mathrm{pre} \rightarrow \mathrm{eff} \rangle$ and consist of a condition pre and a set of effects $\mathrm{eff} = \{\mathrm{eff}_1, \ldots, \mathrm{eff}_n\}$ containing at most one numeric effect for each numeric variable and at most one truth assignment for each propositional variable.

The semantics of a numeric planning task is straightforward. For constants $c \in \mathbb{Q}$, $s(c) = c$. Numeric expressions $(\mathsf{e}_1 \circ \mathsf{e}_2)$ for $\circ \in \{+, -, \times, \div\}$ are recursively evaluated in state s: $s(\mathsf{e}_1 \circ \mathsf{e}_2) = s(\mathsf{e}_1) \circ s(\mathsf{e}_2)$. a numeric constraints $(\mathsf{e}_1 \bowtie \mathsf{e}_2)$, with expressions e_1, e_2 and $\bowtie \in \{\leq, <, =, \neq\}$, $s \vDash (\mathsf{e}_1 \bowtie \mathsf{e}_2)$ is satisfied iff $s(\mathsf{e}_1) \bowtie s(\mathsf{e}_2)$.

Related Work

Extending classical delete relaxation heuristics to numeric problems has been done before, albeit only for a subset of numeric tasks, where numeric variables can only be manipulated in a restricted way. The Metric-FF planning system (Hoffmann 2003) tries to convert the planning task into a *linear numeric* task, which ensures that variables can "grow" in only one direction. When high values of a variable are beneficial to fulfill the preconditions, *decrease* effects are considered harmful. Another approach to solve *linear* numeric planning problems is to encode numeric variables in a linear program and solve constraints with an LP-solver. Coles et al. (2008) analyze the planning problem for consumers and producers of resources to obtain a heuristic that ensures that resources are not more often consumed than produced or initially available. The RANTANPLAN planner (Bofill, Arxer, and Villaret 2015) uses linear programs in the context of planning as satisfiability modulo theories. Instead, we are interested in approaching numeric planning supporting *all* arithmetic base operations.

Contributions

The objective of this dissertation is to provide planning system for numeric planning with instantaneous actions. At the core stands the Fast Downward planning system (Helmert 2006) for classical planning. Our intent is to extended Fast

[1]Parts of this extended abstract were adopted from previous work (Aldinger, Mattmüller, and Göbelbecker 2015)

Downward with full numeric functionality without impairing its performance on classical domains. Numeric Fast Downward (NFD), the numeric extension of Fast Downward has to alter almost all components of the Fast Downward planning systems. Fast Downward transforms the PDDL input into more convenient and effective data structures. During the translation phase, the task is grounded translated into a multi-valued SAS^+ representation. During a knowledge compilation step, domain transition graph and causal graph are determined. Finally, different search algorithms can be used together with a multitude of heuristics in order to solve the planning task. Extending Fast Downward requires major modifications in all steps, and NFD has to deal with interactions between numeric variables, interactions between numeric and multi-valued variables and new challenges coming from the numeric abilities (e.g. an operator can now have infinitely many different outcomes, depending on the previous state). Some of the extensions for NFD could be adopted from Temporal Fast Downward (TFD) (Eyerich, Mattmüller, and Röger 2009), a temporal planner based on an earlier version of Fast Downward, e.g. the handling of numeric expressions by recursively introducing auxiliary variables for each expression. The evaluation of these auxiliary variables is then handled by numeric axioms. Other extensions had to be developed from scratch, either because the Fast Downward evolved from the time when TFD branched from it, or because some features were never implemented for TFD (e.g. the detection of unreachable world states early on can simplify internal data structures).

In order to obtain a baseline heuristic for numeric planning, we implement numeric extensions of the relaxation heuristics from classical planning $h_{\max}$, $h_{\text{add}}$and h_{FF}. In order to do so, we found that the theoretical base for relaxed numeric planning was not set and had to be established first. Numeric planning is undecidable (Helmert 2006) in general, while classical planning is PSPACE-complete (Bylander 1994). Nevertheless, plans exist for many numeric problems and for many numeric problems we can prove unsolvability, so we seek guidance for these problems. This guidance is obtained by heuristics and in order to be tractable we want the estimate to be computable in polynomial time. The idea of a relaxation heuristic is that every fact that is achieved once during planning remains achieved. The problem is simplified as the set of achieved facts grows monotonously. We studied different extensions to relaxation for numeric planning (Aldinger, Mattmüller, and Göbelbecker 2015) and found intervals to be suitable. The idea of an interval relaxation is to store the lower and the upper bound of achievable values in an interval, ensuring that the reachable values can only grow at each step. The methods to deal with intervals have been studied in the field of interval arithmetic. Nevertheless, a major obstacle in numeric planning has to be overcome: the repeated application of numeric operators. While relaxed operators are idempotent in classical planning, the same operator can alter the state of the world arbitrarily often (e.g. $\langle \emptyset \rightarrow \mathsf{x} \mathrel{+}= 1 \rangle$ can increase $x_0 = [0, 0]$ to $x_i = [0, i]$ after i steps). We analyzed conditions under which this repeated application of operators can be captured in polynomial time, and how interval relaxed plans can be derived by explicating the number of repetitions. For *acyclic* numeric planning tasks, i.e. tasks where variables do not depend directly or indirectly on themselves, we proved that the interval relaxation in P. For cyclic tasks, we can introduce cycle breaker actions that artificially set the reachable values of a variable to $(-\infty, \infty)$. While this impairs the quality of the heuristic estimate, it ensures that the heuristic can be computed in polynomial time.

On the practical side we use numeric planning in the context of earth observation satellites application of numeric planning to earth observation satellites (Aldinger and Löhr 2013). An Earth observation satellite equipped with heavy optical sensors has to slew towards regions of interest while orbiting Earth. The number of observation sites exceeds the capability of the satellite and attitude dynamic constraints have to be satisfied.

Open Research Ideas

In the near future we are interested in addressing the open question whether the restriction to acyclic numeric planning tasks can be weakened. We are also interested in tackling another problem inherent to relaxation heuristics: the cyclic resource transfer problem (Coles et al. 2008). Numeric variables are frequently used to model resources. If an operator can transfer resources from one location to another, this is modeled by reducing the quantity at the source location while increasing it at the target. In the relaxed problem, the quantity of the resource is not decreased at the source location, and as such a resource can be "produced" by moving it around. This deteriorates heuristic estimates in many (relevant) numeric planning problems. Coles et al. (2008) use linear programming to ensure that no more resources are consumed than produced. Linear programs are also used in the numeric planning system RANTANPLAN by Bofill, Arxer, and Villaret (2015). We believe that linear programming can be fruitful for numeric planning in many ways and opens many promising research directions for future work.

References

Aldinger, J., and Löhr, J. 2013. Planning for Agile Earth Observation Satellites. In *Proceedings of the ICAPS-2013 Workshop on Planning in Continuous Domains (PCD 2013)*, 9–17.

Aldinger, J.; Mattmüller, R.; and Göbelbecker, M. 2015. Complexity of Interval Relaxed Numeric Planning. In *Proceedings of the 38th German Conference on Artificial Intelligence (KI 2015)*.

Bofill, M.; Arxer, J. E.; and Villaret, M. 2015. The RANTANPLAN Planner: System Description. In *Proceedings of the ICAPS-15 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAPS 2015)*, 1–10.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence 69* 165–204.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. A Hybrid Relaxed Planning Graph-LP Heuristic for Numeric Planning

Domains. In *Proceedings of the 20th International Conference on Automated Planning and Search (ICAPS 2008)*, 52–59.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 130–137.

Fox, M., and Long, D. 2003. `PDDL2.1` : An Extension to `PDDL` for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research 20 (JAIR)* 61–124.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research 26 (JAIR)* 191–246.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating 'Ignoring Delete Lists' to Numeric State Variables. *Journal of Artificial Intelligence Research 20 (JAIR)* 291–341.

Exploiting Search Space Structure in Classical Planning: Analyses and Algorithms (Dissertation Abstract)

Masataro Asai
Graduate School of Arts and Sciences
University of Tokyo

State of the Current Work, Future Plans and Expectations from the Consortium

The author has completed 2 years of research in Masters degree and is in the first year of the PhD, which is not so close to the dissertation. As a result, this dissertation abstract contains several speculative materials. This is because the author's current publications lack the coherent story, primarily due to the lack of good understanding of macro operators and search algorithms in the planning community. I address this issue in the future work sections and make up the coherent story that is necessary to form a viable thesis.

At the time of writing this, I expect from the Consortium the advice how to form a viable, coherent dissertation thesis, which is completely different from writing an individual research paper. I also wish to connect with mentors and students in the Consortium for future collaboration, because some of future work may not make way into the thesis.

In the following sections, I first present my past work, then I propose some future ideas.

Current Work

Factored Planning System CAP

We proposed a Factored Planning framework CAP (Asai and Fukunaga 2015).

Factored Planning (FP) is a class of planning framework which first decompose a problem into (hierarchical) subproblems, then (hierarchically) merge the results of the subproblems into a concrete solution of the entire problem. FP subsumes Hierarchical Task Network in which the decomposition is written by humans. In contrast, recent FP systems use the automatic decomposition of the planning problems (Amir and Engelhardt 2003; Brafman and Domshlak 2006; Kelareva et al. 2007; Fabre et al. 2010).

CAP is a variant of FP systems which only weakly requires the decomposability of the problem. Previous FP systems assume the full *disjointness* (subgoals do not conflict with each other) and the *concatenability* (high-level solver can connect the solutions of the decomposed subproblems), primarily because it tries to solve the problem using *all and only* the solutions to the decomposed subproblems. CAP, in contrast, uses the solutions to the subproblems as macro operators, and compose the plan using macros as well as the primitive actions.

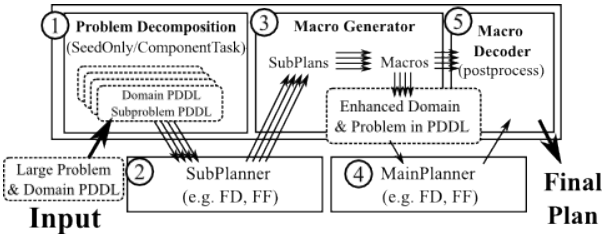


Figure 1: CAP system overview. SubPlanner and MainPlanner are domain-independent planners, e.g., FD/lama (Helmert 2006), FF (Hoffmann and Nebel 2001). They can be the same planner, or different planners.

Figure 1 shows the overview of the CAP framework. SubPlanner and MainPlanner are domain-independent planners, e.g., FD/lama (Helmert 2006), FF (Hoffmann and Nebel 2001), Probe (Lipovetzky and Geffner 2011), YAHSP3 (Vidal and others 2004; Vidal 2011; 2014). They can be the same planner, or different planners (mixed configuration). In detail, CAP works as follows:

1. *Problem Decomposition*: Perform a static analysis of the PDDL problem in order to identify the independent subproblems. Each subproblem is called a *component task*, which is created from an *abstract component*. There are several ways to construct abstract components, which affect the resulting component task.
2. *Generate Subplans with SubPlanner*: Solve the subproblems with a domain-independent planner (SubPlanner).
3. *Macro generation*: For each subplan, concatenate all of its actions into a single, ground (nullary) macro operator.
4. *Main Search by MainPlanner*: Solve the augmented PDDL domain (including macros) with a standard domain-independent planner (MainPlanner).
5. *Decoding*: Finally, any macros in the plan found by MainPlanner are decoded back to the primitive actions.

Unlike the previous Factored Planning frameworks, CAP was shown to be capable of solving wide range of planning problems. We tested CAP in extremely large planning problems generated by the same problem generators in the standard IPC Sequential Satisficing domains, as well as the Learning Track Test instances of IPC. Table 1 shows that CAP and MUM, a state-of-the-art macro learning system, improve performance in

a completely different domains, and that CAP combined with MUM further improves the performance.

		X = FF				X = FD/lama				X = Probe			
	Domain	FF	MUM(FF)	$CAP_s^{7.5}$(FF)	MUM($CAP_s^{7.5}$(FF))	FD/lama	MUM(FD/lama)	$CAP_s^{7.5}$(FD/lama)	MUM($CAP_s^{7.5}$(FD/lama))	Probe	MUM(Probe)	$CAP_s^{7.5}$(Probe)	MUM($CAP_s^{7.5}$(Probe))
IPC2011 Learning	barman-ipc11-learn(30)	0	0	29	**30**	5	0	**29**	0	9	1	24	**30**
	blocksworld-ipc11-learn(30)	6	**25**	6	**25**	25	**29**	25	**29**	19	**29**	20	**29**
	depots-ipc11-learn(30)	2	**3**	1	1	0	0	0	0	28	29	27	**30**
	gripper-ipc11-learn(30)	0	0	0	0	0	**5**	0	**5**	0	**30**	0	**30**
	parking-ipc11-learn(30)	**1**	**1**	**1**	**1**	**14**	**14**	8	10	**4**	2	3	2
	rover-ipc11-learn(30)	2	0	3	**4**	**27**	0	12	23	15	0	10	**19**
	satellite-ipc11-learn(30)	2	1	2	**3**	**5**	0	0	0	0	0	0	0
	spanner-ipc11-learn(30)	0	0	0	0	0	0	0	0	0	0	0	0
	tpp-ipc11-learn(30)	0	9	20	**30**	14	**30**	**30**	**30**	**10**	0	**10**	0
	Sum	13	39	62	**94**	90	78	**104**	97	85	91	94	**140**

Table 1: IPC2011 Learning Track results on 15 minutes, 4GB memory setting, using the standard planner $X \in \{\text{FD/lama, FF, Probe}\}$, with/without either/both of macros introduced by CAP and MUM.

CAP has a plenty of rooms for enhancements. It can be enhanced by using the different planners in the subproblem solving and the main planning enhanced by macros. The timelimit criteria of the subproblem solving can be dynamically optimized by the iterative resource allocation. Some subproblems can be pruned by the compatibility criteria between the subproblems, which is checked by detecting the graph isomorphism. CAP can also be enhanced with a "restoration macro", a macro that "bridges the gap" to the next applicable macro.

Revisiting the Utility Problem: An Empirical Analysis

Although the performance improvement of CAP is clear, we gave further in-depth analysis on why CAP system works and why their enhancements work.

With this task in mind, we revisited the *Utility Problem*, a tradeoff between the benefit and the cost of introducing macros. Although recent macro systems such as MacroFF (Botea et al. 2005), Wizard (Newton et al. 2007) and MUM (Chrpa, Vallati, and McCluskey 2014) employ sophisticated macro pruning methods, some of key assumptions regarding the utility problem predate current heuristic search based planners. We reinvestigate the utility problem for macro operators using two models, "partial solution macros" and "junk macros", each represents how "obviously useful" macros and "obviously useless" macros affect the search performance of planners. As a result, we get the following observations:

First, contrary to conventional wisdom, macro operators do not increase the *effective branching factor* in modern heuristic search-based planners. We show that introducing randomly chosen "junk" macros reduces node evaluations in many domains, and in some domains, junk macros improves the runtime (Table 2).

(LAMA) Domain	L	Preprocess [sec]	Search [sec]	Total [sec]	Eval [node]
airport	8	*112 (1.1)*	**355 (.50)**	**467 (.57)**	**280721 (.74)**
cybersec	8	**2217 (.91)**	3	**2220 (.91)**	3309
depot	8	22 (1.3)	149 (.50)	**171 (.54)**	**190577 (.47)**
driverlog	5	*24 (1.3)*	*105 (1.6)*	*129 (1.5)*	**179752 (.88)**
hanoi	2	3 (1.0)	**287 (.79)**	**290 (.79)**	2070986 (.97)
mystery	5	*87 (1.4)*	**4 (.21)**	*91 (1.1)*	**2643 (.08)**
pipesworld-t	8	*304 (1.5)*	*893 (2.1)*	*1197 (1.9)*	**355576 (.89)**
rovers	2	*331 (1.1)*	114 (.96)	*445 (1.0)*	**87475 (.90)**
transport-sat11	2	*205 (1.3)*	*630 (2.0)*	*835 (1.8)*	**47244 (.47)**

Table 2: Selective results showing the improvements by junk macros of length L, using LAMA planner. Each cell shows the sum over all instances in the domain solved by all configurations, averaged by the 10 runs. Ratios relative to LAMA are shown, e.g., "(.86)" means the ratio compared to LAMA is 0.86. **Improvement**/*degradation* are tested with statistical significance ($p < 0.001$).

	cov.	macros	($L \geq 2$)	usage	(%)	expansion	time
baseline	557	0	0	0	0/0	83009511	1765
split1	561	598	595	557	93.6	16194	0.36
split3	561	1794	1727	1550	89.7	175689	3.74
split10	561	5980	4100	2999	73.1	3683892	50.2
split3gap1	561	1794	1648	1423	86.3	389398	20.6
split3gap3	560	1794	1444	1158	80.2	1811416	74.7
split3gap5	561	1794	1260	984	78.1	7540669	202

Table 3: Results on problems with partial solution macros and partial solution macros with gaps.

Next, we show that the planner may fail to use even trivially useful "partial solution macros".

The most trivially useful macros are the complete solutions to the planning problem itself — Any solution can be encoded as a macro, such that applying it to the initial state results in reaching the goal in one step. Although such macros are clearly unrealistic, understanding the behavior of modern planners with such a macro can yield useful insights.

As a next step we investigate *partial solution macros*, which are the macros generated by splitting a solution into several parts and encoding the individual pieces as macros. Since connecting those macros solve the entire problem instantly, smart planners *should* be able to successfully connect them. We refer to this assumption a *concatenability* assumption, an important assumption made by Factored Planners. However, we empirically show that the planners are in fact not able to connect them, and the concatenability assumption does not hold. We show that an important factor determining such success/failure in utilizing macros is the difficulty of establishing a chain of macro applications, i.e., the "gap" between the partial solution macros (Table 3).

By applying new insights, we can now fully investigate CAP and *restoration macros*, an enhancement to CAP which addresses the problem of large gaps between the macros found by CAP.

Tiebreaking Strategy for A*: How to Explore the Final Frontier

Despite recent improvements in search techniques for cost-optimal classical planning, the exponential growth of the size of the search frontier in A* is unavoidable. We investigate

tiebreaking strategies for A*, experimentally analyzing the performance of standard tiebreaking strategies that break ties according to the heuristic value of the nodes. We find that tiebreaking has a significant impact on search algorithm performance when there are zero-cost operators that induce large plateau regions in the search space. We develop a new framework for tiebreaking based on a depth metric which measures distance from the entrance to the plateau, and proposed a new, randomized strategy which significantly outperforms standard strategies on domains with zero-cost actions (Asai and Fukunaga 2016).

We showed that contrary to conventional wisdom, tiebreaking based on the heuristic value is not necessary to achieve good performance. We also proposed a new framework for defining tiebreaking policies based on *depth*. Our depth-based, randomized strategy $[h, rd, ro]$, which uses the heuristic value, but explicitly avoids depth and ordering biases present in previous methods, significantly outperforms previous strategies on domains with zero-cost actions, including practical application domains with resource optimization objectives in the IPC benchmarks. The proposed approach is highly effective on domains where zero-cost actions create large plateau regions where all nodes have the same f and h costs and the heuristic function provides no useful guidance.

Summary of Contributions

Our current contributions can be summarized as follows. (1) We proposed CAP, a satisficing factored planner using macros. (2) We investigated of the general effect of macro operators in satisficing planning, and applied the new observation to CAP. (3) We investigated the past tiebreaking strategies of A* for optimal search, and proposed a new tiebreaking methods which diversifies the search depth.

Although (1) and (2) are the same line of work, (3) does not nicely fit into the storyline, which will be fixed in the future work as proposed in the following sections.

Introduction (Future Work)

Current State-of-the-Art planner such as Fast Downward (**?**) can solve the planning problems of a moderately large size in a reasonable amount of time, mainly thanks to the greedy forward search combined with sophisticated heuristic functions such as *delete-relaxation* (**?**) and *landmarks* (Richter and Westphal 2010), combined with techniques specifically tailored toward planning problems such as *helpful actions* (**?**) and *consistency criteria* (Lipovetzky and Geffner 2011).

However, Classical Planning is PSPACE-Complete (Bylander 1994) and intractable in general. Above strategies are made upon the assumption that the problems are serially decomposable, and in fact its usefullness does not hold in the random problem instances generated by algorithm A, B or C in (Bylander 1996; Rintanen and others 2004) nor in some domains such as Floortile, Scanalyzer in recent IPCs (Alcázar, Veloso, and Borrajo 2011).

Moreover, there are several satisficing search strategies that seem still yet relatively incompatible to, or independent from the heuristic forward search. Examples include SAT-based planners (Rintanen 2012), Lookaheads (Vidal and others 2004), Macro actions (Chrpa, Vallati, and McCluskey 2015), Factored Planning (Amir and Engelhardt 2003; Asai and Fukunaga 2015), Diversified Search (Imai and Kishimoto 2011; Xie et al. 2014; Burfoot, Pineau, and Dudek 2006).

In our work, we try to provide a consistent theoretical background unifying all these strategies, and then propose several practical algorithms inspired by the new observations.

Macro-conversion of the Search Algorithms

First, we formally define the notion of *best first search with lookaheads (L-BFS)* and show that macro actions can simulate any L-BFS, and vice versa (L-BFS can simulate any macro actions). The intuition is as follows: When BFS starts a depth-first lookahead during the search in a certain condition, that condition can be directly encoded in the preconditions of the macros, although in a problem-specific manner. This unifies various inadmissible search strategies as a modification of the search space using macro actions, which greatly simplifies the discussions in the later sections. We hereafter call the act of simulating L-BFS by macro operators as "macro-conversion".

Phase Transition of the Search Space

Next, we tackles the problem of *Phase Transition* in the complex search space of planning problems. Phase transition in a class of search problems is a phenomenon that the difficulty and the complexity of the problems are ruled by a simple meta-level parameter, and become increasingly easy or hard when the parameter crosses a *critical value*.

In AI research, phase transition was first found in the boolean satisficing problems (Huberman and Hogg 1987; Cheeseman, Kanefsky, and Taylor 1991; Selman, Mitchell, and Levesque 1996) and are recently connected to the physical phenomenon in the Ising model of the spin grass (Barahona 1982). In boolean-SAT problems, the meta-level parameter is the ratio $r = L/N$ of the number of clauses L and the number of propositions N, with a critical value $r_c \approx 4.24$ (Crawford and Auton 1993). In boolean SAT, when $N \to \infty$, the probability of being SAT is 0 when $r < r_c$ and 1 when $r > r_c$. When N is finite, it becomes increasingly difficult to determine the satisfiability when p approaches p_c from either above or below.

In planning problems, previous strategies for analyzing the phase transition are primarily based on the analogy from the boolean satisficing problems. For example, the meta-parameter that is claimed to be controlling the problem difficulty is the ratio of number of operators versus the number of state variables (Rintanen and others 2004). In Algorithm B, (Bylander 1996) A and C (Rintanen and others 2004), planning operators are generated randomly.

We instead analyze the planning problems based on the *Percolation Theory* (Stauffer and Aharony 1994), a theory describing the behavior of the fluid percolating through porous material from one end to the other end. The same theory is already shown to be applied to the pathfinding on random graph and ACO algorithm (Velloso and Roisemberg 2008) because the existence of a satisficing path in a graph is equivalent to percolating the material from one porous site to the goal site with the fluid. However, the search spaces of planning problems and the random graphs are claimed to have the different characteristics (Bylander 1996; Rintanen and others 2004).

Percolation theory dictates that the connectability of the graph is controlled simply by the ratio p of the number of *occupied edges* to the number of all edges. In the infinite graph, the probability p of two points having a path is 0 when the ratio r is below a critical threshold r_c, and is 1 when $r > r_c$. The value of r_c depends on the topology of the graph. In case of finite graph, the probability p becomes a continuous function $p(r)$ which has a *critical region* around r_c where the value grows from 0 to 1. The width of the region is called *correlation index radius*, which basically means the radius in which a node is affected by the other nodes.

Using these theories, we treat the grounded search space directly, rather than through the number of operators. An operator does not represent a single edge in the search space, and instead they representing multiple edges starting from the states which satisfies the precondition — the partial specification of the states. We plan to propose a new random problem generation methods which considers the number of states that is applicable to each operator, and show it achieves a much steeper phase transitions than the previous methods. We will also provide a formal proof that the SAT/UNSAT of the problem approaches to 0/1 around the critical value as the size of the graph approaches to the infinity.

Restart-based, Probabilistically Complete Search Algorithm with Randomly Reduced Number of Edges

We propose a restart-based search algorithms which solve the problems by randomly removing the edges in the search space. The reduced instances may be UNSAT, but we show that as long as we control the number of edges so that the meta-parameter r is above the critical region, we can still solve the problems asymptotically as we restart with different random seeds.

This method has an effect of shifting the meta-level parameter outside the critical region and making the problem easily SAT/UNSAT, which follows the intuitive observation that the search finishes quickly due to the reduced branching factor, or the problem is quickly proven to be UNSAT using reachability analysis on the relaxed planning graph. We plan to empirically show that this method achieves a good performance in IPC problems.

Extensions of Tiebreaking Strategy for A* to Satisficing Planning

We analyse our tiebreaking strategy for A* (Asai and Fukunaga 2016) using macro-conversion and percolation theory. Since the strategy explores the search space sparsely, it would have the similar effect as the previous algorithm (Search Algorithm with Randomly Reduced Number of Edges) on the plateau region of A*.

Using the same tiebreaking strategy, we also propose a constant-error search method as compared to the famous constant-*times*-error method WA*. It divides f-value by a constant error value c, ignoring the remainder. Since it introduces an intensive increase of the plateau region, we use the same tiebreaking strategy as in (Asai and Fukunaga 2016).

Another possible application of this tiebreaking is the temporal planning problems, where the actions with short duration can be hidden behind the actions with longer duration, which is known as ϵ-cost traps (Cushing, Benton, and Kambhampati 2010).

Analysing CAP using Percolation Thoery

Finally, we analyse CAP using Percolation Theory. Since the macros introduced by CAP tends to be long, it has a significant impact on the connectability of the search space. This analysis is expected to finally form a into a coherent story out of the current work which have the different topics.

Conclusion

I summarized several current work of mine (including the materials being under review) and showed that they have diverged topics which are hard to form a coherent thesis. Then I proposed an idea how to merge those topics into a single topic, percolation theory, using the macro-conversion technique.

References

Alcázar, V.; Veloso, M.; and Borrajo, D. 2011. Adapting a rapidly-exploring random tree for automated planning. In *SoCS*.

Amir, E., and Engelhardt, B. 2003. Factored planning. In *IJCAI*, volume 3, 929–935. Citeseer.

Asai, M., and Fukunaga, A. 2015. Solving Large-Scale Planning Problems by Decomposition and Macro Generation. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Asai, M., and Fukunaga, A. 2016. Tiebreaking Strategies for Classical Planning Using A^* Search.

Barahona, F. 1982. On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical and General* 15(10):3241.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *JAIR* 24:581–621.

Brafman, R. I., and Domshlak, C. 2006. Factored planning: How, when, and when not. In *AAAI*, volume 6, 809–814.

Burfoot, D.; Pineau, J.; and Dudek, G. 2006. RRT-Plan: A Randomized Algorithm for STRIPS Planning. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*, 362–365.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence* 69(1):165–204.

Bylander, T. 1996. A Probabilistic Analysis of Prepositional STRIPS Planning. *Artificial Intelligence* 81(1):241–271.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. M. 1991. Where the really hard problems are. In *IJCAI*, volume 91, 331–340.

Chrpa, L.; Vallati, M.; and McCluskey, T. L. 2014. MUM: A Technique for Maximising the Utility of Macro-operators by Constrained Generation and Use. In *ICAPS*, 65–73.

Chrpa, L.; Vallati, M.; and McCluskey, T. L. 2015. On the Online Generation of Effective Macro-Operators. In *International Joint Conference on Artificial Intelligence*, 1544–1550.

Crawford, J. M., and Auton, L. D. 1993. Experimental results on the crossover point in satisfiability problems. In *AAAI*, volume 93, 21–27. Citeseer.

Cushing, W.; Benton, J.; and Kambhampati, S. 2010. Cost Based Search Considered Harmful.

Fabre, E.; Jezequel, L.; Haslum, P.; and Thiébaux, S. 2010. Cost-Optimal Factored Planning: Promises and Pitfalls. In *ICAPS*, 65–72.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. *JAIR* 14:253–302.

Huberman, B. A., and Hogg, T. 1987. Phase transitions in artificial intelligence systems. *Artificial Intelligence* 33(2):155–171.

Imai, T., and Kishimoto, A. 2011. A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning. In *AAAI*.

Kelareva, E.; Buffet, O.; Huang, J.; and Thiébaux, S. 2007. Factored Planning Using Decomposition Trees. In *IJCAI*, 1942–1947.

Lipovetzky, N., and Geffner, H. 2011. Searching for Plans with Carefully Designed Probes. In *ICAPS*.

Newton, M. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning Macro-Actions for Arbitrary Planners and Domains. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*, 256–263.

Richter, S., and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.(JAIR)* 39(1):127–177.

Rintanen, J., et al. 2004. Phase Transitions in Classical Planning: An Experimental Study. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*, volume 2004, 101–110.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artificial Intelligence* 193:45–86.

Selman, B.; Mitchell, D. G.; and Levesque, H. J. 1996. Generating hard satisfiability problems. *Artificial intelligence* 81(1):17–29.

Stauffer, D., and Aharony, A. 1994. *Introduction to percolation theory*. CRC press.

Velloso, B. P., and Roisemberg, M. 2008. Percolation analyses in a swarm based algorithm for shortest-path finding. In *Proceedings of the 2008 ACM symposium on Applied computing*, 1861–1865. ACM.

Vidal, V., et al. 2004. A Lookahead Strategy for Heuristic Search Planning. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*, 150–160.

Vidal, V. 2011. YAHSP2: Keep it simple, stupid. *Angel Garcıa-Olaya, SJ, and López, CL, eds., Seventh International Planning Competition* 83–90.

Vidal, V. 2014. YAHSP3 and YAHSP3-MT in the 8th International Planning Competition. *Vallati, Mauro and Chrpa, Lukáš and McCluskey, Thomas L., Eighth International Planning Competition* 64–65.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-Based Exploration with Multiple Search Queues for Satisficing Planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2395–2402.

UNIVERSITAT DE GIRONA

DEPARTAMENT D'INFORMÀTICA, MATEMÀTICA APLICADA I ESTADÍSTICA
UNIVERSITAT DE GIRONA, SPAIN

Dissertation Abstract: SAT/SMT techniques for planning problems

Author:
Joan ESPASA

Supervisors:
Mateu VILLARET

February 17, 2016

Abstract

Although a lot of work has been devoted to the encoding of planning tasks to propositional logic, only a few works can be found in the literature on satisfiability based approaches to planning in domains that require numeric reasoning. This is probably due to the difficulty of efficiently handling at the same time numeric constraints and propositional formulas. Surprisingly, satisfiability modulo theories (SMT) has been scarcely considered in planning, despite being an active and growing area of research. Since SMT is the natural extension of SAT when propositional formulas need to be combined with numeric constraints, we think it is worth considering SMT for SAT-based planning with numeric domains. The purpose of this thesis is to adapt and take advantage of the expressivity of SMT technology for solving planning problems with numerical constraints. Nevertheless, we remark that most of the results accomplished are generalized to SMT, not just SAT modulo linear arithmetic.

Introduction

The problem of planning, in its most basic form, consists in finding a sequence of actions that will allow to reach a goal state from a given initial state. Although initially considered a deduction problem, it was rapidly seen that it could be better addressed by looking at it as a satisfiability (model finding) problem (Kautz and Selman 1992). Many (incomplete) heuristic methods can be found in the literature to efficiently deal with this problem, most of them oriented towards finding models. Exact methods were ruled out at the beginning due to their inefficiency. However, in (Kautz, McAllester, and Selman 1996) it was shown that modern off-the-shelf SAT solvers could be effectively used to solve planning problems. In the last years, the power of SAT technology has been leveraged to planning (Rintanen 2012), making reduction into SAT state of the art for deterministic planning.

As we have stated, a lot of work has been devoted to the encoding of plans in propositional logic, but only a few works can be found in the literature on satisfiability based approaches to planning in domains that require numeric reasoning. However, the advances in satisfiability modulo theories (SMT) (Barrett et al. 2009) in the last years make worth considering this alternative.

The pioneering work of LPSAT (Wolfman and Weld 1999) on planning with resources can indeed be considered one of the precursors of SMT, as the basic ideas of SMT (Boolean abstraction, interaction of a SAT solver with a theory solver, etc.) were already present in it. A comparison between SAT and SMT based encodings for planning in numeric domains can be found in (Hoffmann et al. 2007).

Other approaches, related to SMT to some amount as well, have been developed more recently. In (Belouaer and Maris 2012), a set of encoding rules is defined for spatio-temporal planning, taking SMT as the target formalism. On the other hand, in (Gregory et al. 2012) a modular framework, inspired in the architecture of lazy SMT, is developed for planning with resources.

Hipothesis

Methods and techniques for SAT have been adapted and extended successfully for dealing with problems modelled using more expressive logics than propositional. In the case of SAT Modulo Theories (SMT), its objective is to decide the satisfiability of a set of propositional formulas, in combination of theories like equality, linear or real integer arithmetic, bit vectors, ... The application of SMT technology to combinatorial problems has given very good results. The purpose of this thesis is to adapt and take advantage of the expressivity of SMT technology for solving planning problems with numerical constraints. This objective can be seen as a two step plan: Finding good encodings for the problems, and then adapting the current solvers to tackle planning problems more efficiently.

Efficient Encodings

The first step we have focused on is to efficiently encode PDDL problems into SMT problems.

As SMT is an expressive language, a first translation of planning problems expressed with PDDL was achieved, wich resulted in a publication (Bofill, Espasa, and Villaret 2014) where it was shown that it has promising results.

Our approximation is competitive with other exact and complete methods for planning with resources on the tackled problem, and also with some incomplete (heuristic) ones. In particular, we obtained better results than NumReach (Hoffmann et al. 2007) and similar results to SGPlan (Hsu and Wah 2008).

But some of the instances were big or long enough to make this approach not feasible. As the number of variables, and hence the search space, rapidly grows with the number of time steps considered, a key idea to improve the performance of this approach is to consider the possibility of executing several actions at the same time, i.e., the notion of parallel plans. Parallel plans increase the efficiency not only because they allow to reduce the time horizon, but also because it is unnecessary to consider all total orderings of the actions that are performed in parallel.

Parallelization of actions relies in the notion of (non-)interference, which is usually determined at compile time, i.e., independently of any state. This method often overestimates the chances of interference, but guarantees the feasibility of the plan.

To ensure that a parallel plan is sound, it is necessary that all actions proposed to be executed at the same time do not interfere. Different notions of interference have been defined, some more restrictive, some more relaxed.

A generalization of the standard notion of interference between actions in SAT-based planning to the case of formulas over Boolean and linear arithmetic propositions, makes it suitable for planning with resources. In particular, we are developing novel ideas that can help to determine in a more fine-grained way interference between actions, as we think that it is a key ingredient for dealing with planning with resources efficiently.

To illustrate the situations where classic notions of interference(Fox and Long 2003) overestimate affectation be-

tween actions, consider the following example. The problem consists in transporting people between cities using cars. Each car has a limited number of seats and a given fuel capacity. The actions on this example are `drive` and `board`.

A car can only drive if it is transporting somebody and it has enough fuel to reach its destination, and boarding is limited by seat availability:

```
(:action fly
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :precondition (and (at ?a ?c1)
                    (> (onboard ?a) 0)
                    (>= (fuel ?a)
                    (distance ?c1 ?c2)))
 :effect (and (not (at ?a ?c1))
              (at ?a ?c2)
              (decrease (fuel ?a) (distance ?c1 ?c2))))

(:action board
 :parameters (?p - person ?a - aircraft ?c - city)
 :precondition (and (at ?p ?c)
                    (at ?a ?c)
                    (> (seats ?a) (onboard ?a)))
 :effect (and (not (at ?p ?c))
              (in ?p ?a)
              (increase (onboard ?a) 1)))
```

The classic notion of interference would determine interference between `drive` and `board`, since `board` modifies the `onboard` function (number of passengers) and `drive` checks the value of this function in its precondition. We are developing techniques that can find out that there is no interference at all, since it is impossible that the preconditions of `board` and `drive` were true at the same time, and after executing `board` the precondition of `drive` becomes false. Note that the precondition of `drive` requires `(> (onboard ?a) 0)` and the effect `(increase (onboard ?a) 1)` of `board` can never falsify `(> (onboard ?a) 0)`.

Parallelism in the Planning Modulo Theories framework

Planning Modulo Theories (PMT) (Gregory et al. 2012) is a modular framework that generalizes the integration of arbitrary theories with propositional planning. It is inspired in the architecture of lazy SMT, which is the natural extension of SAT when propositional formulas need to be combined with other theories.

Existing works on numeric planning use syntactic or limited semantic approaches to determine interference between actions, in a fairly restrictive way (Kautz and Walser 1999; Fox and Long 2003; Gerevini, Saetti, and Serina 2008), and not much is said in terms of interference between actions when other theories are involved

Following the advances we made regarding parallelism in the field of planning with resources, we decided to generalize our ideas to a more general framework like PMT. We accomplished a more general notion of interference between actions, new relaxed semantics for parallel plans and a chained encoding that can benefit from these, all in the context of PMT.

This chained encoding, as its name suggests, lets the solver chain the effects of various actions that the notion classifies as non-interfering in one time-step.

Actually we are working in a relaxed version of our notion of interference for PMT, together with new encodings that can benefit from it. We aim to decrease further the number of necessary checks to reach a valid plan.

More compact encodings

As the previously introduced encodings grow considerably with time, to the point of getting unmanageable instances in some big domains, we are developing more compact encodings, using the theory of uninterpreted functions to express predicates, functions and actions. These encodings are reminiscent of the lifted causal encodings in (Kautz, McAllester, and Selman 1996).

In the SMT-LIB standard (Barrett, Stump, and Tinelli 2010), QF_UFLIA stands for the logic of Quantifier-Free Boolean formulas, with Linear Integer Arithmetic constraints and Uninterpreted Functions. Uninterpreted functions have no other property than its name and arity. In other words, they are only subject to the following axiom schema of consistency: $x_1 = x'_1 \wedge \cdots \wedge x_n = x'_n \implies f(x_1, \ldots, x_n) = f(x'_1, \ldots, x'_n)$.

Every defined object (ship, port, cargo, ...) in the problem is mapped to an integer. For each function, predicate and action an uninterpreted function is declared. Uninterpreted functions corresponding to predicates and actions return a Boolean value, whilst the ones for functions return an integer value.

This encoding is more compact, and it retains most of the problem original structure. It remains to be seen if a parallelized version of this encoding could lead to better results than the encoding without functions. To the best of our knowledge, there are no works using parallelized encodings with uninterpreted functions. It should hence be studied how to generalize the standard parallel encodings to this setting.

Adapting SMT solvers

All the abovementioned encodings and techniques are being implemented in the RANTANPLAN planner (Bofill, Espasa, and Villaret 2015). To this day, existing works to solve combinatorial problems using SMT are based in using SMT solvers as black boxes. This approximation has obvious limits, as much is left to the internal design decisions of the solver strategies.

We aim to adapt SMT solvers for efficiently modelled planning problems, helping the solver search strategies and even adapting existing theories to our benefit. There is a lot of room for improvement in this area, and we expect to be able to continue doing meaningful contributions to the community.

References

Barrett, C.; Sebastiani, R.; Seshia, S.; and Tinelli, C. 2009. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, volume 185. IOS Press. chapter 26, 825–885.

Barrett, C.; Stump, A.; and Tinelli, C. 2010. The Satisfiability Modulo Theories Library (SMT-LIB). `http://www.SMT-LIB.org`.

Belouaer, L., and Maris, F. 2012. SMT Spatio-Temporal Planning. In *ICAPS 2012 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS 2012)*, 6–15.

Bofill, M.; Espasa, J.; and Villaret, M. 2014. Efficient SMT Encodings for the Petrobras Domain.

Bofill, M.; Espasa, J.; and Villaret, M. 2015. The RANTANPLAN Planner: System Description. *Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS-15)* 1.

Fox, M., and Long, D. 2003. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)* 20:61–124.

Gerevini, A. E.; Saetti, A.; and Serina, I. 2008. An approach to efficient planning with numerical fluents and multicriteria plan quality. *Artificial Intelligence* 172(8):899–944.

Gregory, P.; Long, D.; Fox, M.; and Beck, J. C. 2012. Planning Modulo Theories: Extending the Planning Paradigm. In *Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI.

Hoffmann, J.; Gomes, C. P.; Selman, B.; and Kautz, H. A. 2007. SAT Encodings of State-Space Reachability Problems in Numeric Domains. In *20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 1918–1923.

Hsu, C.-W., and Wah, B. W. 2008. The SGPlan Planning System in IPC-6. `http://wah.cse.cuhk.edu.hk/wah/Wah/papers/C168/C168.pdf`.

Kautz, H., and Selman, B. 1992. Planning as Satisfiability. In *10th European Conference on Artificial Intelligence (ECAI 92)*, 359–363. John Wiley & Sons, Inc.

Kautz, H., and Walser, J. P. 1999. State-space planning by integer optimization. In *AAAI/IAAI*, 526–533.

Kautz, H. A.; McAllester, D. A.; and Selman, B. 1996. Encoding Plans in Propositional Logic. In *Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR 96)*, 374–384. Morgan Kaufmann.

Rintanen, J. 2012. Planning as Satisfiability: Heuristics. *Artificial Intelligence* 193:45–86.

Wolfman, S. A., and Weld, D. S. 1999. The LPSAT Engine & Its Application to Resource Planning. In *Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 99)*, 310–317. Morgan Kaufmann.

Session 5

Planning under Uncertainty and Applications

Robotic control through model-free reinforcement learning

Hofer Ludovic

1 Introduction

My PhD thesis aims at developping new reinforcement learning algorithms specifically designed to control model-free stochastic systems. This thesis is supervised by Hugo Gimbert and Olivier Ly from Bordeaux University, at the LaBRI. The main targeted application is learning of bipedal walking for low-cost humanoid robots, the experimental platform used is Sigmaban (Passault et al. 2015). Modeling such a task properly involves taking into account the backlash of the reduction gears, the bending of the parts as well as the contact with the ground. Therefore, model-based approaches are not suited to learn such a task. We chose to model this task as a Continuous State and Action Markov Decision Process (CSA-MDP). Since it is hard to predict the shape of the optimal policy, we use value iteration method based on the q-value.

It has already been exhibited that RL algorithms could bring improvements for dynamical tasks requiring a very high accuracy such as the ball-in-a-cup task (Kober and Peters 2009) or hitting a baseball (Peters and Schaal 2008). However most of the reinforcement learning involving robots are based on policy gradient method. While those methods are very effective, they present two major drawbacks: the motor primitive has to be defined by the user and they require the possibility of computing the gradient of the reward with respect to the parameters of the motor primitive. Therefore, applying those methods require a high repeatability of the system and they are limited to a family of solutions specified as a parameter of the algorithm.

Due to the lack of repeatability in low-cost robotics systems, it is quite common to represent them as CSA-MDP. This field has known major breakthrough recently, such as the possibility to find exact solutions when the model is known and has discrete noise, piecewise linear transitions and piecewise linear reward (Zamani, Sanner, and Fang 2012), based on the use of symbolic dynamic programming and extended algebraic decision diagrams (Sanner, Delgado, and de Barros 2012). Although these theorical results are outstanding, they cannot be used to control low-cost robots because of the requirements on the transition and reward functions.

Among the previous work on model-free solvers for MDP, we can note the Least-Square Policy Iteration (LSPI) algorithm (Lagoudakis 2003) which manage to learn hard tasks on problems with continuous state and discrete actions. Although this algorithm lead to satisfying results, it requires expert function approximators adapted to the problem. On the other hand, Fitted Q-Iteration (FQI) (Ernst, Geurts, and Wehenkel 2005) grows regression forests from gathered samples and achieve slightly lower performance than LSPI without requiring custom function approximators. While both methods were initially designed for discrete action choices, Binary Action Search (BAS) (Pazis and Lagoudakis 2009) allows to use them on CSA-MDP.

In order to apply algorithms such as LSPI or FQI to high-dimensional control problems, it is mandatory to use efficient exploration algorithm in order to reduce the number of samples required to learn a near-optimal policy. Optimistic algorithms such as Multi-Resolution Exploration (MRE) (Nouri and Littman 2009) allows to improve the process of collecting samples, while providing guarantees to converge to a nearly-optimal solution.

2 Tools and Methods

As mentioned previously, there is a gap between model-free CSA-MDP methods and robotic applications. Complex tasks such as bipedal walking involves high-dimensional spaces for state and actions. Therefore, it is hard to predict the time required to converge to a near-optimal strategy. Moreover, running experiments directly on robots require human supervision and the manufacturing process is costly and time consuming. In order to run realistic simulations and to make our source code more easy to use, we decided to use ROS[1] and Gazebo[2]. Once learning algorithms lead to satisfying results in simulation, it will be possible to test them directly on the robots.

[1] http://www.ros.org

[2] http://www.gazebosim.org

We base our learning of the q-value on the FQI algorithm (Ernst, Geurts, and Wehenkel 2005), which uses regression forests. While our current implementation of regression forest is based on Extra-Trees (Geurts, Ernst, and Wehenkel 2006), we also plan to test and develop other algorithms growing regression forests.

We compute an approximation of the greedy policy corresponding to the q-value calculated by FQI algorithm using regression forest. This process provides two advantages: firstly, it allows to retrieve actions at a very low computational cost, secondly, by smoothing the discretization noise on the q-value, it also improve the performance of the controler. Real-time constraint is particularly important to ensure that closed-loop control is available.

Currently, exploration is ensured by an algorithm based on MRE (Nouri and Littman 2009). This algorithm is based on the optimistic approach which considers the the couple state-actions which are unknown lead to a maximal reward. It allows to build a knowledge function based on kd-trees, this function provides a result in $[0, 1]$, which is mainly based on the ratio between the density inside the leaf and inside the whole tree. Using this information, collected samples are modified in order to increase the reward if they use an unknown transition or lead to an unknown state. In order to obtain a smoother value function, we use a forest of kd-trees.

While MRE update the policy at a fixed interval of step (chosen by the user), this method leads to an increasing time of update, even if the time required to compute the policy grows linearly with respect to the number of samples, the time required to collect n samples grow quadratically with respect to n. On the other hand, if we wait too many steps before updating the policy, there is a high risk of getting stucked in attracting trajectories. In this case, the collected samples will be redundant and will not improve quickly the knowledge of the MDP. Moreover, on real robots, all the updates to the policy have to be quick enough to ensure that the control frequency can be maintained. In other words, there are no ways to freeze the system in its state. In order to face this issue we plan to develop an algorithm allowing to insert dynamically new samples without needing to restart the learning process from scratch. Another option allowing to increase the space between two consecutive update of the policy would be to detect attracting trajectories.

Another issue relative to solving CSA-MDP is long-term reward. This problem is particularly strong for FQI, since each iteration on the value update involves an approximation. While some problems such as stabilizing an inverted pendulum are harder when the control frequency is lower, a very high frequency can make intractable problems such as inverted pendulum swing-up since it would require to compute the q-value at a very high horizon. We plan to test the effect of including the time during which an action should be applied as one of the dimension of the action.

References

Ernst, D.; Geurts, P.; and Wehenkel, L. 2005. Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research* 6(1):503–556.

Geurts, P.; Ernst, D.; and Wehenkel, L. 2006. Extremely randomized trees. *Machine Learning* 63(1):3–42.

Kober, J., and Peters, J. 2009. Policy Search for Motor Primitives in Robotics. *Advances in Neural Information Processing Systems 21* 849–856.

Lagoudakis, M. 2003. Least-squares policy iteration. *The Journal of Machine Learning Research* 4:1107–1149.

Nouri, A., and Littman, M. L. 2009. Multi-resolution Exploration in Continuous Spaces. *Advances in Neural Information Processing Systems* 1209–1216.

Passault, G.; Rouxel, Q.; Hofer, L.; Guyen, S. N.; and Ly, O. 2015. Low-cost force sensors for small size humanoid robot. 33405.

Pazis, J., and Lagoudakis, M. G. 2009. Binary action search for learning continuous-action control policies. *Proceedings of the 26th International Conference on Machine Learning (ICML)* 793–800.

Peters, J., and Schaal, S. 2008. Reinforcement learning of motor skills with policy gradients. *Neural Networks* 21(4):682–697.

Sanner, S.; Delgado, K. V.; and de Barros, L. N. 2012. Symbolic Dynamic Programming for Discrete and Continuous State MDPs. In *Proceedings of the 26th Conference on Artificial Intelligence*, volume 2.

Zamani, Z.; Sanner, S.; and Fang, C. 2012. Symbolic Dynamic Programming for Continuous State and Action MDPs.

Dissertation Abstract: Exploiting Symmetries in Sequential Decision Making under Uncertainty

Ankit Anand
Indian Institute of Technology, Delhi
New Delhi,India-110016
ankit.anand@cse.iitd.ac.in

Synopsis

The problem of sequential decision making under uncertainty, often modeled as an MDP is an important problem in planning and reinforcement learning communities. Traditional MDP solvers operate in flat state space and don't scale well in large state and action spaces. A lot of real world domains have exponential number of states in terms of representation but many of these states and actions are symmetric to each other. In this work, we focus on exploiting symmetry in these domains to make contemporary algorithms more efficient and scalable. Our recent works ASAP-UCT and OGA-UCT which define new state-action pair symmetries and apply them in UCT show promising initial results of this approach. We study important research questions related to finding and using symmetry based abstractions and discuss interesting links with lifted inference in graphical models.

Introduction

The problem of sequential decision making under uncertainty, often modeled as a Markov Decision Process (MDP), is a fundamental problem in the design of autonomous agents(Russell and Norvig 2003). Traditional MDP solving algorithms (value iteration and variants) perform offline dynamic programming or linear programming in flat state spaces and scale poorly with the number of domain features due to the curse of dimensionality. A well-known approach to reduce computation in these scenarios is through domain abstractions. An interesting aspect which have been observed in many domains of interest is that even though flat state space is very large, many states are symmetric to one other. Existing offline abstraction techniques (Givan, Dean, and Greig 2003; Ravindran and Barto 2004) make use of these symmetries and compute equivalence classes of states such that all states in an equivalence class have the same value. This projects the original MDP computation onto an abstract MDP, which is typically of a much smaller size. We intend to study these symmetry exploiting abstractions in traditional MDP setup as well as state of art algorithms which are mostly online, anytime and deal with very large state and action spaces.

Our recent works (Anand et al. 2015b)(Anand et al. 2015a), expands the aforesaid traditional notion of symmetries by giving a novel notion of abstractions, *state-action pair* (SAP) abstractions, where in addition to computing equivalence classes of states, we also compute equivalence classes of state-action pairs, such that Q-values of state-action pairs in the same equivalence class are the same. Moreover, SAP abstractions find symmetries even when there aren't many available state abstractions, which is commonly true for many domains in practice.

During the last decade, Monte-Carlo Tree Search (MCTS) algorithms have become quite an attractive alternative to traditional approaches. MCTS algorithms, exemplified by the well-known UCT algorithm (Kocsis and Szepesvári 2006), intelligently sample parts of the search tree in an online fashion. They can be stopped anytime and usually return a good next action. A UCT-based MDP solver (Keller and Eyerich 2012) won the last two probabilistic planning competitions (Sanner and Yoon 2011; Grzes, Hoey, and Sanner 2014). Unfortunately, UCT builds search trees in the original flat state space too, which is wasteful if there are useful symmetries and abstractions in the domain.

One of the recent works which correct this limitation is by (Jiang, Singh, and Lewis 2014), which introduced the first algorithm to combine UCT with automatically computed approximate state abstractions, and showed its value through quality gains for a single deterministic domain. Our preliminary experiments with this method (which we name AS-UCT: Abstractions of state in UCT) on probabilistic planning domains indicate that it is not as effective in practice. This may be because AS-UCT tries to compute state abstractions on the explored part of the UCT tree and there likely isn't enough information in the sampled trees to compute meaningful state abstractions. In our recent works (Anand et al. 2015a)(Anand et al. 2016), we fill this gap by implementing SAP abstractions inside the UCT framework.

In our recent work (Anand et al. 2015a), we develop an algorithm- ASAP-UCT(**A**bstraction of **S**tate-**A**ction **P**airs in **UCT**) which is a first attempt to exploit SAP abstractions. ASAP-UCT is a batch algorithm like AS-UCT and alternates between two phases. One phase consists of an abstraction computation routine that uses the existing UCT tree to induce groups of symmetric nodes. These nodes are aggregated to construct an abstract search tree. The second phase

is the (modified) UCT algorithm, which is run as per original UCT in the beginning, but is modified to incorporate the abstractions after the abstraction routine has been run at least once. Experiments show that ASAP-UCT significantly outperforms both AS-UCT and vanilla UCT on a number of planning domains obtaining upto 26% performance improvements.

Our further research shows that these batch algorithms do not achieve the full potential of abstractions because of the two disjoint phases. Since abstractions are computed on a sampled tree, they are approximate. Erroneous abstractions computed as part of one batch of abstraction computation may get corrected only after a full phase of modified UCT – this wait could severely impact the solution quality.

In response, we propose *On the Go Abstractions* (OGA), a novel approach in which abstraction computation is tightly integrated into the MCTS algorithm in our recent work (Anand et al. 2016). We implement these on top of UCT and name the resulting algorithm OGA-UCT. It has several desirable properties, including (1) rapid use of new information in modifying existing abstractions, (2) elimination of the expensive batch abstraction computation phase, and (3) focusing abstraction computation on important part of the sampled search space. We experimentally compare OGA-UCT against ASAP-UCT, a recent state-of-the-art MDP algorithm as well as vanilla UCT algorithm. We find that OGA-UCT is robust across a suite of planning competition and other MDP domains, and obtains up to 18 % quality improvements. Based on these initial promising results, we intend to develop a domain independent state of art planner which can benefit from domain abstractions.

Lastly, in the past decade, there has also been revival of interest in abstractions and symmetries with the emergence of lifting techniques in probabilistic graphical models literature. Many of the problems which previously were intractable in probabilistic inference can now be solved by these advances in lifting techniques. Also, there is a strong co-relation between MDP solving methods and probabilistic inference as both of these algorithms depend upon local interactions between neighboring nodes. The ideas of Counting BP (Kersting, Ahmadi, and Natarajan 2009) is very similar to block-splitting algorithm proposed by Givan et. al. Also, homomorphisms (Bui, Huynh, and Riedel 2012) have also been well studied in probabilistic inference literature. Under this theme, we would like to explore the possibility of unifying these two different problems and other related problems under a common abstraction framework so that a generic abstraction approach for solving these can be developed.

Background and Related Work

An infinite horizon, discounted cost Markov Decision Process(MDP) (Puterman 1994) is modeled as a 5-tuple $(S, A, \mathcal{T}, C, \gamma)$. An agent in a state $s \in S$ executes an action $a \in A$ making a transition to $s' \in S$ with a probability $\mathcal{T}(s, a, s')$ incurring a cost $C(s, a)$ with a discount factor of γ ($\gamma < 1$). A policy $\pi : S \rightarrow A$ specifies an action to be executed in a state $s \in S$. Given a starting state $s_0 \in S$, the expected discounted cost $V^\pi(s)$ associated with a policy π is given by $V^\pi(s^0) = E[\sum_{t=0}^{\infty} C(s^t, a^t)\gamma^t | \pi(s^t) = a^t, t \geq 0]$ where expectation is taken over the transition probability $\mathcal{T}(s^t, a^t, s^{t+1})$ of going from state s^t to s^{t+1} under action a^t. The expected cost $Q^\pi(s, a)$ denotes the discounted cost of first taking action a in state s and then following π from then on. The optimal policy π^* minimizes the total expected cost for every state $s \in S$, i.e. $\pi^*(s) = argmin_\pi V^\pi(s)$. $Q^*(s, a)$ and $V^*(s)$ are shorthand notations for $Q^{\pi^*}(s, a)$ and $V^{\pi^*}(s)$ respectively, and $V^*(s) = \min_{a \in A} Q^*(s, a)$. Presence of goals can be dealt by having absorbing states for goals.

An MDP can be equivalently represented as an AND-OR graph (Mausam and Kolobov 2012) in which OR nodes are MDP states and AND-nodes represent state-action pairs whose outgoing edges are multiple probabilistic outcomes of taking the action in that state. Value Iteration (Bellman 1957) and other dynamic programming MDP algorithms can be seen as message passing in the AND-OR graph where AND and OR nodes iteratively update Q(s,a) and V(s) (respectively) until convergence.

A finite-horizon MDP executes for a fixed number of steps (horizon) and minimizes expected cost (or maximizes expected reward). States for this MDP are (s, t) pairs where s is a world state and t is number of actions taken so far. Finite horizon MDPs can be seen as a special case of infinite horizon MDPs by having all the states at the horizon be absorbing goal states and setting $\gamma = 1$.

Abstractions for Offline MDP Algorithms

In many MDP domains, several states behave identically, and hence, can be abstracted out. Existing literature defines abstractions via an equivalence relation $\mathcal{E} \subseteq S \times S$, such that if $(s, s') \in \mathcal{E}$, then their state transitions are equivalent (for all actions). All states in an equivalence class can be collapsed into a single aggregate state in an abstract MDP, leading to significant reductions in computation.

Various definitions for computing abstractions exist. Givan et al. (2003)'s conditions deduce two states to have an equivalence relation if they have the same applicable actions, local transitions lead to equivalent states and immediate costs are the same. Ravindran and Barto (2004) refine this by allowing the applicable actions to be different as long as they can be mapped to each other for this state pair. This can find more state abstractions than Givan's conditions. We call these settings AS (Abstractions of States) and ASAM (Abstractions of States with Action Mappings), respectively.

Our framework ASAP unifies and extends these previous notions of abstractions – we go beyond just an equivalence relation $\mathcal{E}$ over states, and compute equivalences of *state-action pairs*. This additional notion of abstractions leads to a discovery of many more symmetries and obtains significant computational savings when applied to online algorithms.

Monte-Carlo Tree Search (MCTS)

Traditional offline MDP algorithms store the whole state space in memory and scale poorly with number of domain features. Sampling-based MCTS algorithms offer an attractive alternative. They solve finite-horizon MDPs in

an online manner by interleaving planning and execution steps. A popular variant is UCT (Kocsis and Szepesvári 2006), in which during the planning phase, starting from the root state, an expectimin tree is constructed based on sampled trajectories. At each iteration, the tree is expanded by adding a leaf node. Since these MDPs are finite horizon a node is (state,depth) pair. UCT chooses an action a in a state s at depth d based on the UCB rule, $argmin_{a \in A} \left(Q(s,d,a) - K \times \sqrt{\frac{log(n(s,d))}{n(s,d,a)}} \right)$ where $K > 0$. Here, $n(s,d)$ denotes the number of trajectories that pass through the node (s,d) and $n(s,d,a)$ is the number of trajectories that take action a in (s,d).

Evaluation of a leaf node is done via a random *rollout*, in which actions are randomly chosen based on some default rollout policy until a goal or some planning horizon P is reached. This rollout results in an estimate of the Q-value at the leaf node. Finally, this Q-value is backed up from the leaf to the root. UCT operates in an anytime fashion – whenever it needs to execute an action it stops planning and picks the best action at the root node based on the current Q-values. The planning phase is then repeated again from the newly transitioned node. Due to the clever balancing of the exploration-exploitation tradeoff, MCTS algorithms can be quite effective and have been shown to have significantly better performance in many domains of practical interest(Gelly and Silver 2011).

Abstractions for UCT

Hostetler et. al. (2014) develop a theoretical framework for defining a series of state abstractions in sampling-based algorithms for MDP. But they do not provide any automated algorithm to compute the abstractions themselves. Closest to our works is (Jiang, Singh, and Lewis 2014), which applies Givan's definitions of state abstractions within UCT. The key insight is that instead of an offline abstraction algorithm, they test abstractions only for the states enumerated by UCT. Since UCT solves finite-horizon MDPs, only the states at the same depth will be considered equivalent. Then, at any given depth, they test Givan's conditions (transition and cost equality) on pairs of states to identify ones that are in the same equivalence class. This algorithm proceeds bottom-up starting from last depth all the way to the root. Their paper experimented on a single deterministic game playing domain and its general applicability to planning was not tested. We advance Jiang's ideas by applying our novel SAP abstractions in UCT, and show that they are more effective on a variety of domains.

ASAP: Abstraction of State-Action Pairs

In this section, we introduce a new type of State-Action Pair (SAP) abstractions (proposed by us) in addition to previously defined State Abstractions. SAP abstractions are general and can be used independently by any of MDP solving algorithms.

Our **Abstractions of State-Action Pairs** (ASAP) framework unifies and extends Givan's and Ravindran's definitions for computing abstractions. To formally define the framework we introduce some notation. Consider an MDP $M = (S, A, \mathcal{T}, C, \gamma)$. We use P to denote the set of state-action pairs i.e. $P = S \times A$. We define an equivalence relation $\mathcal{E}$ over pairs of states i.e. $\mathcal{E} \subseteq S \times S$. Let $\mathcal{X}$ denote the set of equivalence classes under the relation $\mathcal{E}$ and let $\mu_{\mathcal{E}} : S \rightarrow \mathcal{X}$ denote the corresponding equivalence function mapping each state to the corresponding equivalence class. Similarly, we define an equivalence relation $\mathcal{H}$ over pairs of SAPs i.e. $\mathcal{H} \subseteq P \times P$). Let $\mathcal{U}$ denote the set of equivalences classes under the relation $\mathcal{H}$, and let $\mu_{\mathcal{H}} : P \rightarrow \mathcal{U}$ denote the corresponding equivalence function mapping state-action pairs to the corresponding equivalence classes. Next, we will recursively define state equivalences over state-pair equivalences and vice-versa.

State Abstractions: Suppose we are given SAP abstractions, and $\mu_{\mathcal{H}}$. Intuitively, for state equivalence to hold, there should be a correspondence between applicable actions in the two states such that the respective state-action pair nodes are equivalent. Formally, let $a, a' \in A$ denote two actions applicable in s and s', respectively. We say that two states s and s' are equivalent to each other (i.e, $\mu_{\mathcal{E}}(s) = \mu_{\mathcal{E}}(s')$) if for every action a applicable in s, there is an action a' applicable in s' (and vice-versa) such that $\mu_{\mathcal{H}}(s,a) = \mu_{\mathcal{H}}(s',a')$.

SAP Abstractions: As in the case of state abstractions, assume we are given state abstractions and the $\mu_{\mathcal{E}}$ function. Two state-action pairs (s,a), $(s',a') \in P$ are said to be equivalent (i.e. $\mu_{\mathcal{H}}(s,a) = \mu_{\mathcal{H}}(s',a')$ if:

- $\forall s_i \in S$ such that $\mathcal{T}(s,a,s_i) = p$, $\exists s'_i \in S, \mu_{\mathcal{E}}(s_i) = \mu_{\mathcal{E}}(s'_i)$ and $\mathcal{T}(s',a',s'_i) = p$. (Condition 1(a))
- $\forall s'_i \in S$ such that $\mathcal{T}(s',a',s'_i) = p$, $\exists s_i \in S, \mu_{\mathcal{E}}(s'_i) = \mu_{\mathcal{E}}(s_i)$ and $\mathcal{T}(s,a,s_i) = p$. (Condition 1(b))
- $C(s,a) = C(s',a')$ (Condition 2)

Intuitively, for state-action pair equivalence to hold, the corresponding states that they transition to should be equivalent and the respective transition probabilities should match. Second condition requires the costs of applying corresponding actions to be identical to each other. For Goal-directed MDPs, all goal states are in an equivalence class: $\forall s, s' \in G, \mu_{\mathcal{E}}(s) = \mu_{\mathcal{E}}(s')$. For finite-horizon MDPs, all goal states at a given depth are equivalent.

Example: Figure 1 illustrates the AND-OR graph abstractions on a soccer domain. Here, four players wish to score a goal. The central player (S0) can pass the ball left, right or shoot at the goal straight. The top player (S1) can hit the ball right to shoot the goal. Two players at the bottom (S2, S3) can hit the ball left for a goal. The equivalent AND-OR graph for this domain is the leftmost graph in the figure. Givan's Abstraction of States (AS) conditions check for exact action equivalence. They will observe that S2 and S3 are redundant players and merge the two states. Ravindran's Abstraction of States with Action Mapping (ASAM) conditions will additionally look for mappings of actions. They will deduce that S1's right is equivalent to S2's left and will merge these two states (and actions) too. They will also notice that S0's left and right are equivalent. Finally, our ASAP framework will additionally recognize that S0's straight is equivalent to S1's right and merge these two SAP nodes.

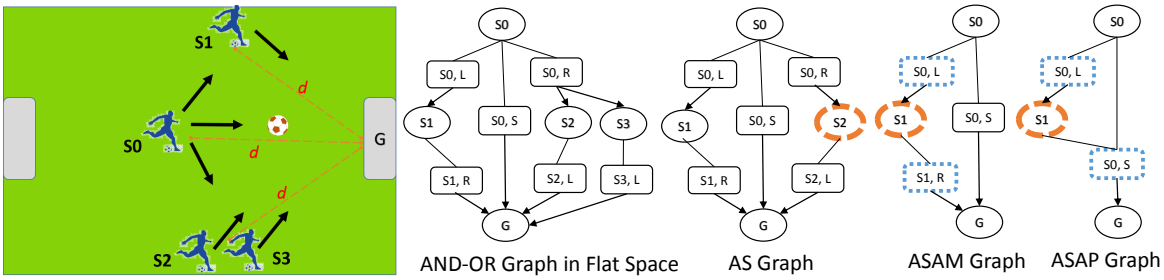


Figure 1: An example showing abstractions generated by various algorithms on a soccer domain. Givan's AS, Ravindran's ASAM and our ASAP frameworks successively discover more and more symmetries.

Overall, ASAP will identify the maximum symmetries in the problem. Next, we state theoretical results corresponding to ASAP.

Theorem 1. *Both AS and ASAM are special cases of ASAP framework. ASAP will find all abstractions computed by AS and ASAM.*

Theorem 2. *Optimal value functions $V^*_{Gr}(x)$, $Q^*_{Gr}(x,u)$, computed by Value Iteration on a reduced AND-OR graph Gr, return optimal value functions for the original MDP M. Formally, $V^*_{Gr}(x) = V^*_M(s)$, and $Q^*_{Gr}(x,u) = Q^*_M(s,a)$, $\forall s \in S,\ a \in A$ s.t. $\mu_{\mathcal{E}}(s) = x, \mu_{\mathcal{H}}(a) = u$.*

ASAP Symmetries in UCT

We next describe algorithms to incorporate ASAP framework in UCT. Firstly, we describe a batch algorithm ASAP-UCT which is followed by OGA-UCT. OGA-UCT builds on ASAP-UCT by computing abstractions on the go as we are building the tree. We show empirically that ASAP-UCT performs better than AS-UCT and ASAM-UCT and further illustrate that OGA-UCT outperforms ASAP-UCT on several domains of interest.

ASAP-UCT

ASAP-UCT is a UCT-based algorithm that uses the power of abstractions computed via the ASAP framework. Recall that since UCT constructs a finite-horizon MDP tree, states at different depths have to be treated differently. Therefore, ASAP-UCT tests state equivalences for states at the same depth only. To compute abstractions over UCT tree, we adapt and extend ideas in Jiang et al. (2014)'s work.

Computing UCT Abstractions: ASAP-UCT computes abstractions in a bottom up fashion starting with the leaves and successively computing abstractions at each level (depth) all the way to the root (Algorithm 1). It takes as input a UCT Search Tree (ST) and outputs an Abstracted Search Tree (AST). At each level, it calls the functions for computing state and state-action pair abstractions, alternately.

For the pseudo-code it is helpful to understand each depth as consisting of a layer of state nodes and a layer of SAP nodes above it. We use the superscript d to denote the state (SAP) pair equivalence function $\mu^d_{\mathcal{E}}$ ($\mu^d_{\mathcal{H}}$) at depth d. Similarly, we use S^d to denote the set of states at depth d and P^d to denote the set of SAP nodes at depth d. To keep the notation simple, we overload the equivalence function (map) $\mu^d_{\mathcal{E}}$ to also represent the actual equivalence relationship over state pairs (similarly for $\mu^d_{\mathcal{H}}$). $ComputeAS$ and $ComputeASAP$ at each level operate as per the definitions described in ASAP framework. As we are going bottom up, the abstractions at below level have already been computed.

Algorithm 1 Computing Abstracted Search Tree

ComputeAbstractedSearchTree(SearchTree ST)
  $d_{max} \leftarrow$ getMaxDepth(ST), $\mu^{d_{max}+1}_{\mathcal{H}} \leftarrow \{\}$
  for $d := d_{max} \rightarrow 1$ **do**
    $\mu^d_{\mathcal{E}} \leftarrow$ ComputeAS(S^d, $\mu^{d+1}_{\mathcal{H}}$);
    $\mu^d_{\mathcal{H}} \leftarrow$ ComputeASAP(P^d, $\mu^d_{\mathcal{E}}$);
  end for
  $AST \leftarrow$ SearchTree with Computed Abstractions
  return AST

Updating Q-Values: Once the nodes at a level have been made a part of the same abstract state, we maintain an estimate of the expected cost (to reach the goal state) for the abstract node only (both in the state layer as well as in the state-action layer). The abstract node is initialized with the average of the expected cost of its constituent in the beginning. Any future Q-value updates are performed over the abstracted out representation.

When to Compute Abstractions: Since we need the current sampled tree for calculating the abstractions, abstraction can be computed only after the tree has been constructed to a certain level. But if we wait until the full expansion of the tree (i.e. end of the planning phase), the abstractions would not be useful. We compute abstractions for a fixed number of times l in each decision. After every abstraction, the Q-values are computed on the abstract tree. Future expansions might invalidate the abstractions computed earlier. We correct for this by performing the next phase of abstractions from scratch on the flat (unabstracted) tree. In summary, the algorithm can be described as a batch algorithm which interleaves expansions, Q-value computations and abstraction steps.

Experimental Results with ASAP-UCT We compare the four algorithms, vanilla UCT, AS-UCT, ASAM-UCT and ASAP-UCT, in all three domains. For each domain instance we vary the total time per trial and plot the average cost obtained over 1000 trials. Figures 2 shows the comparisons across two domains. Note that time taken for a trial also includes the time taken to compute the abstractions. In almost

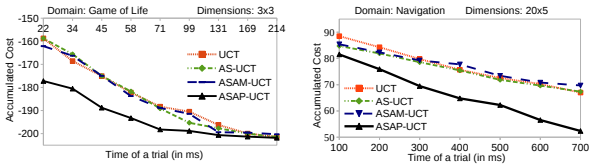


Figure 2: ASAP-UCT outperforms all other algorithms on problems from three domains.

all settings ASAP-UCT vastly outperforms both UCT, AS-UCT and ASAM-UCT. ASAP-UCT obtains dramatically better solution qualities given very low trial times incurring up to 26% less cost compared to UCT. Its overall benefit reduces as the total trial time increases, but almost always it continues to stay better or at par.

OGA-UCT

Here, we describe OGA-UCT, an On the Go Abstraction algorithm which computes abstractions as we are building the tree. Our algorithm is best understood in terms of the construction of the original UCT tree. The UCT computation can be broadly divided in three phases 1) Sampling of a trajectory 2) Random rollout from a newly discovered leaf node 3) Back up of Q-values. In OGA-UCT, during the first phase, along with sampling of the trajectory, an abstraction for each state is also maintained *on the go*. Abstraction for any node is computed using the recursive updates similar to the ones used by ASAP-UCT. But the key difference is that instead of doing the batch computation uniformly for each node, we do it incrementally and in an adaptive manner. Each node has an associated *r*ecency count which stores the number of times the node was visited after its abstraction was last updated. If the recency count reaches a pre-decided threshold K, we re-compute the abstraction for this node and set the recency count back to 0. In the second phase when a rollout is performed, we initialize the abstraction of the newly created leaf and set its recency count to 0. Any Q-value updates in the UCT tree are now done over the abstract nodes rather than the original nodes. Since abstractions at a certain depth depend on the abstractions in the tree below, it may happen that when a node's abstraction changes, there could be a change in the abstraction of its ancestor nodes. Therefore, any change in the abstraction of a node at depth d, is propagated all the way up to the root of the tree, recomputing abstractions as necessary. We describe the Sampling Trajectory Procedure 2 in detail here.

Sampling Trajectory (Algorithm 2): This is the main procedure of our algorithm. Lines 1-4 check the base condition for stopping a trajectory. Lines 7-11 add a newly discovered leaf node to the tree, initialize its abstraction and perform a rollout. If the procedure comes to line 12, we have not discovered a new leaf node yet. Line 12 selects an action based on the UCB rule. In lines 13-14, we add a newly discovered SAP node to the tree and initialize its abstraction. Lines 18-19 sample a new state node based on the chosen action and recursively call SAMPLETRAJECTORY. Lines 19-23 take care of maintaining the recency count and calling update abstractions if the count has reached the threshold K. Here, Update SAP abstraction updates the abstractions with respect to the state abstractions at the next level. Also, if the abstraction of SAP node changes, this change calls Update State Abstractions for the parent node and this update procedure is recursively repeated till the root if the abstractions changes. Finally, the UCT counts and Q values are updated in lines 24-26. It is insightful to note that if we remove the lines for computing abstractions and maintaining the recency count (lines 9,15-16,20-23), the procedure becomes identical to what standard UCT would do.

Algorithm 2 Sample Trajectory in UCT

1: **procedure** VAL = SAMPLETRAJECTORY(s, d)
2: **if** terminal(s) **then**
3: **return** $-reward(s)$
4: **else if** $d ==$ $Horizon$ **then**
5: **return** 0
6: **end if**
7: **if** (s, d) is not in tree T **then**
8: Add state node (s, d) to tree T
9: INITIALIZESTATEABSTRACTION(s, d)
10: **return** GETROLLOUT(s, d)
11: **end if**
12: $a \leftarrow$ SELECT-UCB-ACTION(s, d)
13: **if** (s, a, d) is not in tree T **then**
14: Add SAP node (s, a, d) to tree T
15: INITIALIZE-SAP-ABSTRACTION(s, a, d)
16: $RecencyCount[s, a, d] \leftarrow 0$
17: **end if**
18: $s' \leftarrow$ SAMPLE(s, a)
19: $RecencyCount[s, a, d] + +$
20: $newVal \leftarrow$ SAMPLETRAJECTORY$(s', d + 1)$
21: **if** $RecencyCount[s, a, d] == K$ **then**
22: UPDATE-SAP-ABSTRACTION(s, a, d)
23: **end if**
24: INCREMENTCOUNT(s, a, d)
25: UPDATEQ$((s, a, d), newVal)$
26: **return** $newVal$
27: **end procedure**

It is also important to note that OGA-UCT converges to correct Q-values as computed by UCT given sufficiently large amount of time.

Theorem 3. *Given an MDP $M = (S, A, \mathcal{T}, C, H)$, the value function computed by OGA-UCT for the abstract node containing a state s at depth d converges to the value function computed by UCT for state s, as number of trajectories $N \to \infty$ i.e $\forall s \in S \;\; \forall d \leq Horizon$*

$$\lim_{N\to\infty} V^N_{OGA}(\mu^d_{\mathcal{X}}(s), d) = \lim_{N\to\infty} V^N_{UCT}(s, d)$$

Empirical Results of OGA-UCT We compare OGA-UCT with ASAP-UCT and unabstracted UCT on these problems with different total planning times and draw cost vs. time curves. Representative runs on two domains are illustrated in Figure 3. Each curve is an average of 1,000 reruns and 95% confidence interval bars are also drawn.

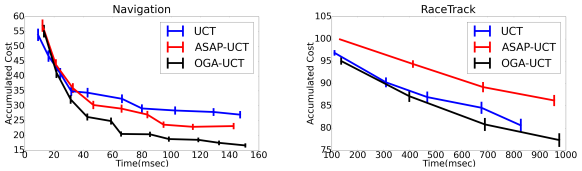


Figure 3: OGA-UCT performs better or at par with ASAP-UCT and UCT for most of the domains

We observe that OGA-UCT performs the best or on par with the best on four out of the five domains. These results demonstrate that difference in the performances of ASAP-UCT and UCT can depend heavily on the domain, but OGA-UCT admits least variance and is robust across these many domains.

Proposed Scope and Future Focus

Abstraction and symmetry in MDPs in itself is a relatively old field with rich theoretical literature on it. This work has assumed great importance in today's world with the need for real time MDP solvers for large problem instances. The advent of space exploration missions like Mars rover is a perfect example of reinforcement learning problem where hard multiple objectives need to be achieved with in constraints of time, cost and safety. We plan to extend these abstraction frameworks and adapt these in complete end to end systems which can be used in real world. To achieve this, we will focus on some or all of these problems.

- **A Domain Independent Abstraction Based Planner:** Initial investigations of applying abstractions in UCT have shown promising results both in OGA-UCT and ASAP-UCT. Presently both OGA-UCT and ASAP-UCT operate in flat state space, how to modify and use the factored representations is an important step in the development of such a planner.
- **Learning Abstractions:** With advances in machine learning techniques, an interesting approach to compute abstractions is by learning the symmetries of state space. A recent work by Srinivasan et. al. (Srinivasan, Talvitie, and Bowling 2015) suggests the use of nearest neighbor approach to improve exploration in UCT. Learning abstractions is an important problem to be studied in context of online algorithms where abstraction computation overhead may become a bottleneck.
- **Relation between Lifted inference and Planning** As pointed out earlier, symmetries have played a significant role in advancing inference techniques in graphical models. Due to local nature of computation, there is significant overlap of techniques used to exploit symmetries in both planning and graphical models. We intend to study this co-relation in detail and wish to develop a generic abstraction framework for both these fields.

Finally, we believe that exploiting symmetry based abstractions could lead to significant improvements in many algorithms not limited to planning and reinforcement learning. Our initial investigations with it and development of ASAP-UCT and OGA-UCT clearly show the first step in this direction. Nevertheless, there are significant challenges like computing symmetries efficiently, operating in factored state space and adapting comtemporary algorithms to compute symmetries which are non-trivial and need to be investigated thoroughly.

References

Anand, A.; Grover, A.; Mausam; and Singla, P. 2015a. ASAP-UCT: Abstraction of State-Action Pairs in UCT. In *IJCAI*, 1509–1515.

Anand, A.; Grover, A.; Mausam; and Singla, P. 2015b. A Novel Abstraction Framework for Online Planning. In *AAMAS*.

Anand, A.; Noothigattu, R.; Mausam; and Singla, P. 2016. OGA-UCT: On-the-Go Abstractions in UCT. In *ICAPS*.

Bellman, R. 1957. A Markovian Decision Process. *Indiana University Mathematics Journal*.

Bui, H. H.; Huynh, T. N.; and Riedel, S. 2012. Automorphism groups of graphical models and lifted variational inference. *CoRR* abs/1207.4814.

Gelly, S., and Silver, D. 2011. Monte-carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence* 175(11):1856–1875.

Givan, R.; Dean, T.; and Greig, M. 2003. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence* 147(1–2):163 – 223.

Grzes, M.; Hoey, J.; and Sanner, S. 2014. International Probabilistic Planning Competition (IPPC) 2014. In *ICAPS*.

Hostetler, J.; Fern, A.; and Dietterich, T. 2014. State Aggregation in Monte Carlo Tree Search. In *AAAI*.

Jiang, N.; Singh, S.; and Lewis, R. 2014. Improving UCT Planning via Approximate Homomorphisms. In *AAMAS*.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic Planning Based on UCT. In *ICAPS*.

Kersting, K.; Ahmadi, B.; and Natarajan, S. 2009. Counting Belief Propagation. In *UAI*, UAI '09, 277–284. Arlington, Virginia, United States: AUAI Press.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML*. Springer.

Mausam, and Kolobov, A. 2012. *Planning with Markov Decision Processes: An AI Perspective*. Morgan & Claypool Publishers.

Puterman, M. 1994. *Markov Decision Processes*. John Wiley & Sons, Inc.

Ravindran, B., and Barto, A. 2004. Approximate homomorphisms: A framework for nonexact minimization in Markov decision processes. In *Int. Conf. Knowledge-Based Computer Systems*.

Russell, S. J., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

Sanner, S., and Yoon, S. 2011. International Probabilistic Planning Competition (IPPC) 2011. In *ICAPS*.

Srinivasan, S.; Talvitie, E.; and Bowling, M. 2015. Improving Exploration in UCT Using Local Manifolds. In *AAAI Conference on Artificial Intelligence*.

Recommending and Planning Trip Itineraries for Individual Travellers and Groups of Tourists

Kwan Hui Lim[*†]
[*]Department of Computing and Information Systems, The University of Melbourne, Australia
[†]Victoria Research Laboratory, National ICT Australia, Australia
limk2@student.unimelb.edu.au

Abstract

Trip planning is both challenging and tedious for tourists due to their unique interest preferences and various trip constraints. Despite the availability of online resources for tour planning and services provided by tour agencies, there are various challenges such as: (i) selecting POIs that are personalized to the unique interests of individual travellers; (ii) constructing these POIs as an itinerary, with considerations for time availability and starting/ending place preferences (e.g., near a tourist's hotel); (iii) for tour agencies to group tourists into tour groups such that the recommended tour appeals to the interests of the group as a whole; and (iv) similarly, for tour agencies to assign tour guides with the right expertise to lead each of these tour groups. In our work, we aim to develop algorithms for recommending personalized tours to both individual travellers and groups of tourists, based on their interest preferences, which we automatically determine based on geo-tagged photos posted by these tourists. Using a Flickr dataset of geo-tagged photos as ground-truth for real-life POI visits in multiple cities, we evaluate our proposed algorithms using various metrics such as precision, recall, F1-score, user interest scores and POI popularity, among others.

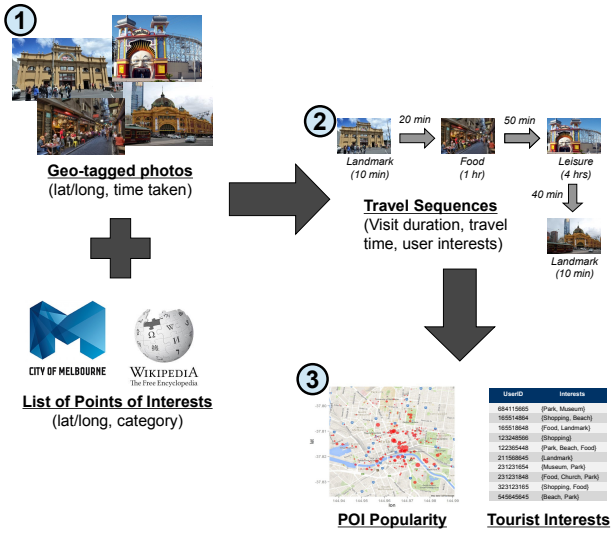


Figure 1: Tour Recommendation Framework. The various steps indicate: (1) Mapping of geo-tagged photos to list of POIs; (2) Construction of tourist visit history/sequences; and (3) Calculation of POI popularity and tourist interests.

1 Introduction

1.1 Motivations

Tourism is an important industry to the world economy, contributing more than US$1.2 trillion in revenue and accounting for more than 1.1 billion international tourists (UNWTO 2015). Despite the importance of tourism, planning a tour or trip itinerary is still a challenging task for any visitor in a foreign city, due to unfamiliarity with the various Points of Interest (POI) in the city. Although there are many online resources available for tour planning, there still exist challenges such as: (i) many travel guides simply recommend popular POIs that do not reflect the tourist's interest preferences or consider various trip constraints, such as the available time for touring and preferred starting/ending location, e.g., starting and ending near the tourist's accommodation; and (ii) even after obtaining a list of POIs, it is a tedious task to construct an itinerary of sequential POI visits with the considerations of travelling time, visiting time, and specific starting/ending points.

One possible solution is to engage the services of tour operators to organize such tour itineraries. However, tour operators may not be aware of the unique interests of individual tourists and face the same challenge of recommending tours that are personalized to the tourist's interest preferences. Furthermore, tour operators typically offer group tours to multiple tourists and face the additional challenges of: (i) optimizing for an appropriate tour group size, e.g., large groups to minimize cost overheads or small groups to maximize tourist experience; (ii) constructing tours with POIs that are appealing to multiple tourists in a group; and (iii) assigning tour guides with the appropriate expertise to best lead each tour group.

Our work aims to address the challenges of recommend-

ing tours that are suitable for individual travellers[1] and groups of tourists, in particular, considering the diverse set of interests among these tourists. To achieve these goals, we implemented a tour recommendation framework (Figure 1) that utilizes geo-tagged photos (Flickr) and crowd-sourced information (Wikipedia), and proposed various algorithms based on variants of the Orienteering problem and various clustering algorithms. In the following section, we describe the three main research questions that we aim to address as part of this work.

1.2 Research Questions

Our PhD research aims to develop tour recommendation algorithms that are meaningful at multiple levels, namely for individual tourists, groups of tourists, and the entire tourist population. Our work is further motivated by the following research questions (RQ):

- **RQ 1:** At the individual level, how can we recommend personalized tours that consider the interest preferences and trip constraints (e.g., time/distance budget and preferred starting/destination POIs) of individual tourists?
- **RQ 2:** At the group level, how can we recommend group tours that consider appropriate tour group sizes, interest preferences for multiple tourists in a group and assignment of tour guides based on their expertise?
- **RQ 3:** At the global level, how can we recommend tours that benefit the tourist population as a whole? I.e., how do we plan tours that minimizes undesirable effects at POIs, such as over-crowdedness and long queuing times?

1.3 Related Work

As our work aims to recommend tours for individual travellers and groups of tourists, we first discuss some state-of-the-art works in the respective areas of tour recommendation for individual travellers and tour recommendation for groups of tourists.

Tour Recommendation for Individuals. There are various works that aim to recommend tours for individuals, i.e., a single tourist, and we discuss some key literature from this area. Many of these works approach tour recommendation as an optimization problem, such as the Orienteering problem (Tsiligirides 1984; Vansteenwegen, Souffriau, and Oudheusden 2011) or Generalized Maximum Coverage problem (Cohen and Katzir 2008). For example, (Choudhury et al. 2010) was one such work that recommended tours for an individual tourist, with a specific starting and ending POI, while ensuring that the tour can be completed within a certain time. Others like (Gionis et al. 2014) extended upon this research area by implementing the constraint of a sequence ordering to the POI visits, e.g., restaurant $\rightarrow$ shopping $\rightarrow$ beach $\rightarrow$ park. Similarly, (Brilhante et al. 2013; 2015) modelled tour recommendation based on the Generalized Maximum Coverage problem, with considerations for both POI popularity and user interests. Other tour recommendation research also included transportation-related considerations, such as (Chen et al. 2015) that considered varying travelling times based on traffic conditions, and (Kurashima et al. 2010; 2013) that utilized different modes of transportation in their travel routes. For more information, (Gavalas et al. 2014) provides a comprehensive discussion of algorithms that aim to recommend tours to individual tourists. In addition, there have been many web/mobile-based applications developed for the same purpose such as (Brilhante et al. 2014; Refanidis et al. 2014; Castillo et al. 2008), which are based on variations of the discussed works.

Tour Recommendation for Groups. In recent years, group recommendations have been studied in-depth by researchers, such as by (Amer-Yahia et al. 2009) and (Hu et al. 2014), who proposed and applied group recommendation algorithms to the retail domain, i.e., recommending top-k retail items such as movies, books, music.[2] For the tourism domain, there are many interesting works that apply group recommendation algorithms for tourism-related purposes, resulting in applications such as *e-Tourism* (Garcia, Sebastia, and Onaindia 2011; Garcia et al. 2009), *Intrigue* (Ardissono et al. 2003) and *Travel Decision Forum* (Jameson, Baldes, and Kleinbauer 2003). Extending upon (Sebastia et al. 2009), *e-Tourism* (Garcia, Sebastia, and Onaindia 2011; Garcia et al. 2009) explicitly solicits the interest preferences and group membership details of users, then recommends tours that best satisfy the interest preferences of the entire group based on the user-provided groupings. Other applications like *Intrigue* (Ardissono et al. 2003) and *Travel Decision Forum* (Jameson, Baldes, and Kleinbauer 2003) aim to fulfil a similar purpose of recommending tours to groups of tourists. The main difference is that *Intrigue* requires users to provide their POI preferences instead of specific interests, while *Travel Decision Forum* includes an additional online discussion phase to get its users to mutually agree on proposed changes to the tour itinerary.

1.4 Structure and Organization

The rest of the paper is organized as follows. In Section 2, we describe some of our main contributions in the area of tour recommendation. In Section 3, we discuss some future directions that we aim to embark on for the remaining of my PhD. Finally, we summarize and conclude this paper in Section 4.

[1] We use the terms "traveller" and "tourist" interchangeably.

[2] While group recommendation research is related to tour recommendation for groups of tourists, the latter involves additional challenges, such as constructing the recommended POIs (items) into a connected itinerary and considerations for specific starting/ending points, and a limited time budget for visiting and travelling between POIs. As such, we focus more on literature regarding tour recommendation for groups of tourists and refer readers to (Boratto and Carta 2011) for a more comprehensive discussion on group recommendation works.

2 Contributions to Date

In the following sections, we describe some of our main contributions thus far, which include: (i) implementing a general framework for deriving user-POI visit history based on geo-tagged photos (Section 2.1); (ii) formulating the basic tour recommendation problem and various variants (Section 2.2); (iii) proposing tour recommendation algorithms for individual tourists (Section 2.3); and (iv) proposing tour recommendation algorithms for groups of tourists (Section 2.4).

2.1 General Framework

As illustrated in Figure 1, our overall tour recommendation framework makes use of: (i) geo-tagged photos that are tagged with a geographical coordinates (latitude/longitude) and stamped with the time taken; and (ii) a POI list comprising POI names, category and latitude/longitude coordinates. The geo-tagged photos can be obtained from any photo sharing website such as Flickr or Instagram, and the POI list can be obtained from Wikipedia or a specific city's tourism-related website (e.g., City of Melbourne). This framework comprises the following steps:

1. Map geo-tagged photos to a list of POIs if their coordinates differ by a specific distance, e.g., $\leq$100m, resulting in a list of POI visits. For calculating this spherical (earth) distance, we make use of the Haversine formula (Sinnott 1984).

2. Construct the tourist travel history by connecting POI visits (obtained from Step 1) of the same tourist. In particular, we derive a user's visit duration at a POI based on the time difference between his/her first and last photo (of a consecutive nature) at that POI.

3. Calculate POI popularity and tourist interest preferences based on tourist travel histories from Step 2. POI popularity is based on the number of visits to a specific POI (the more visits, the more popular), while tourist interest is based on variations of POI visit durations (which is discussed later).

While Step 1 typically uses geo-tagged photos, it can be easily extended to other media with a lat/long coordinate and time-stamp, e.g., GPS traces on mobile phones or other location-based social networking services such as geo-tagged tweets on Twitter. As input to this framework, we use Wikipedia and Flickr geo-tagged photos that are publicly available as part of the Yahoo! Flickr Creative Commons 100M dataset (Yahoo! Webscope 2014; Thomee et al. 2016).[3] This framework was used in various of our works, such as (Lim et al. 2015b; Lim 2015; Lim et al. 2016), which we describe in more detail in the later sections.

[3]Our pre-processed dataset (i.e., photos mapped to POI visits and visit sequences) are also made publicly available at https://sites.google.com/site/limkwanhui/datacode.

2.2 Basic Problem Definition

We now restate the basic tour recommendation problem definition that we described in (Lim et al. 2015b). Given the set of POIs P, a budget B, starting POI $p_1 \in P$, destination POI $p_N \in P$, our main goal is to recommend a tour itinerary that maximizes both user interests $Int(Cat_i)$ and POI popularity $Pop(i)$, while adhering to the budget B. Formally, we want to construct a tour itinerary $I = (p_1, ..., p_N)$ that:

$$Max \sum_{i=2}^{N-1} \sum_{j=2}^{N} x_{i,j} \Big(\eta Int(Cat_i) + (1-\eta) Pop(i) \Big) \quad (1)$$

where $x_{i,j} = 1$ if we travel directly from POI i to j (i.e., we visit POI i, followed by POI j), and $x_{i,j} = 0$ otherwise. We then attempt to solve for Eqn. 1, subjected to the following constraints:

$$\sum_{j=2}^{N} x_{1,j} = \sum_{i=1}^{N-1} x_{i,N} = 1 \quad (2)$$

$$\sum_{i=1}^{N-1} x_{i,k} = \sum_{j=2}^{N} x_{k,j} \leq 1, \qquad \forall\, k = 2, ..., N-1 \quad (3)$$

$$\sum_{i=1}^{N-1} \sum_{j=2}^{N} Cost(i,j) x_{i,j} \leq B \quad (4)$$

$$2 \leq p_i \leq N, \qquad \forall\, i = 2, ..., N \quad (5)$$

$$p_i - p_j + 1 \leq (N-1)(1 - x_{i,j}), \qquad \forall\, i, j = 2, ..., N \quad (6)$$

Eqn. 1 attempts to maximize a dual-objective of POI popularity and user interests on all POIs in the recommended tour itinerary, and η controls the emphasis given to either POI popularity or user interests. Constraints 2 to 6 ensures that: (i) the itinerary starts and ends at POI 1 and N, respectively (Constraint 2); (ii) all POIs in the itinerary are connected and no POIs are re-visited (Constraint 3); (iii) the total time taken to visit all POIs in the itinerary is within the time budget B, based on a function $Cost(p_x, p_y)$ that is computed from both a personalized POI visit duration and travelling time between POIs (Constraint 4); (iv) there are no sub-tours (separate self-looping tours) in the proposed solution, based on the sub-tour elimination constraint proposed in (Miller, Tucker, and Zemlin 1960) for the Travelling Salesman Problem (Constraints 5 and 6). We then proceed to solve this tour recommendation problem as an integer programming problem and for this purpose, we used the lpsolve linear programming package (Berkelaar, Eikland, and Notebaert 2004).

2.3 Tour Recommendation for Individual Tourist

In the first year of my PhD, we focused our research on personalized tour recommendation for individual tourists (RQ 1). In this research area, there has been various works that aim to recommend interest-based tours based on the Generalized Maximum Coverage problem (Brilhante et al. 2015) and using a combination of topic and Markov models (Kurashima et al. 2013). We built upon these earlier works by exploring an intuitive model of user interests based on POI visit time and recommending tour itineraries with a mandatory visit category. Our contributions include:

Tour Recommendation with Personalized POIs and Visit Duration. In (Lim et al. 2015b), we proposed the PERSTOUR algorithm for recommending personalized tours with POIs and visit duration based on POI popularity and time-based user interests. This algorithm models POI popularity based on POI visit count, and time-based user interests using a tourist's total visit duration at POIs of a certain category, relative to that of an average tourist. Our intuition is that a tourist is more interested in a POI category if he/she spends more time at POIs of this category. For determining tourist POI visit duration, we utilize the geo-tagged photos taken by a user and calculate their POI visit duration based on the time difference between the first and last photo taken at a specific POI. Based on measures of tour popularity, tourist interest, recall, precision and F1-score, experimental results show that our PERSTOUR algorithm is able to recommend POIs and visit durations that more accurately reflect tourists' real-life visits, compared to various greedy-based baselines. For more information on this work, please refer to (Lim et al. 2015b).

Customized Tour Recommendation with Mandatory Categories. In (Lim 2015), we proposed the TOURRECINT algorithm for recommending customized tours with a mandatory POI category based on tourist interests. This algorithm optimizes a variant of the Orienteering problem (Tsiligirides 1984; Vansteenwegen, Souffriau, and Oudheusden 2011), with a time/distance budget, starting POI, destination POI and mandatory POI category. We defined this mandatory POI category as the most frequently visited POI category based on a tourist's visit history. Thereafter, we solve this variant of the Orienteering problem as an integer programming problem. Using a ground truth of real-life POI visits by tourists (based on their geo-tagged photos), experimental results show that TOURRECINT out-perform various baselines, in terms of precision, recall and F1-score. For more information on this work, please refer to (Lim 2015).

2.4 Tour Recommendation for Groups of Tourists

In the second year of my PhD, we proceeded to investigate customized tour recommendation for groups of tourists (RQ 2). While there is extensive literature on group recommendation of top-k items (Boratto and Carta 2011) and tour recommendation for individual tourist (Gavalas et al. 2014), there is limited work on tour recommendation for groups of tourists. For works that explore tour recommendation for groups (Garcia, Sebastia, and Onaindia 2011; Ardissono et al. 2003; Jameson, Baldes, and Kleinbauer 2003), they focus more on the group recommendation aspect and do not consider the assignment of tour guides to lead these tour groups. Similarly, many of these works assume that the tourist groupings and interest preferences are explicitly provided. As part of RQ 2, we aim to study tour recommendation for groups as a more holistic problem, which includes grouping tourists with diverse interest preferences, recommending tour itineraries and assigning tour guides to these groups. Our contributions include:

Group Tour Recommendation with Tour Guide Assignment.[4] In (Lim et al. 2016), we introduced the Group Tour Recommendation (GROUPTOURREC) problem, which involves recommending tours that best satisfy the interest preferences of groups of tourists, where each tour group is subsequently led by a tour guide. To solve this GROUPTOURREC problem, we proposed an approach for recommending group tours that aims to: (i) determine tourist interests based on past POI visits, and cluster tourists with similar interests into a group; (ii) recommend tours to groups based on a variant of the Orienteering problem that considers both group interests and POI popularity; and (iii) assigns tour guides with the appropriate expertise to lead each tour, using an integer programming approach. In addition, this problem is also technically challenging due to its NP-hard complexity. As such, we use greedy-based approaches and integer programming to solve for smaller sub-problems of tourists grouping, POI recommendation and tour guides assignment, as part of the group tour recommendation problem. Based on various measures of group interest similarity, total/maximum/minimum tour interests and total tour guide expertise, results show that our proposed approach out-performs various baselines, including standard tour packages offered by real-life tour agencies. For more information on this work, please refer to (Lim et al. 2016).

Detecting Location-centric Communities. In (Lim et al. 2015a), we investigated a complementary problem of detecting communities of users that frequently visit or reside in similar locations. In this work, we proposed the use of Social-Spatial-Temporal (SST) links, which are traditional social/friendship links between users augmented with spatial and temporal information, e.g., visited the same place within a certain time-frame. Using standard community detection algorithms (such as the Louvain (Blondel et al. 2008), Infomap (Rosvall and Bergstrom 2008) and LabelProp (Raghavan, Albert, and Kumara 2007) algorithms) on these SST links, we were able to detect location-centric communities comprising users who exhibit strong similarities in terms of the places they visit and reside in. In another related work (Lim and Datta 2016), we also observed that user communities with similar interests are more likely to reside in the same locality. In the future, we intend to extend this work to determine if users are travelling alone or as a group, and accordingly recommend tour that are appropri-

[4] This work (Lim et al. 2016) will also be presented at the 26th International Conference on Automated Planning and Scheduling (ICAPS'16).

ate for individuals and groups. For more information on this work, please refer to (Lim et al. 2015a).

3 Future Research Plan

For the remaining of my PhD, we aim to work on tour recommendation strategies that benefit the tourist population as a whole (RQ 3) and we intend to work on the following:

Game-theoretic Approaches to Tour Recommendation. Traditionally, tour recommendation algorithms aim to propose tours that maximize the personal profit of individual tourists. One limitation of this approach is that while the individual tourist benefits, the entire tourist population could potentially "lose" (e.g., everyone going to the most popular POI but ends up overcrowding and creating long queues at that POI, leading to a poor tour experience for most people). To address this problem, we intend to adopt a game theoretic approach to tour recommendations where we model POI "crowdedness" as a common utility and derive equilibrium strategies to recommend tours that will benefit all tourists as a whole. Potential applications of this work would be in optimizing for queuing times at attractions/rides in theme parks and preventing over-crowding at exhibits within museums. For example, instead of recommending the most popular attraction in a theme park to all visitors and increasing the queuing times, we may want to recommend some less popular attractions that have shorter queuing times to a subset of visitors.

4 Conclusion

In summary, we introduced the general problem of tour recommendation, and discussed in greater detail, the specific problems of recommending tours for individual travellers and groups of tourists, along with the consideration of their unique interest preferences. We then described our various contributions in the general area of tour recommendation, which include the following:

- Proposing the PERSTOUR algorithm for recommending personalized tours with POIs and visit duration based on POI popularity and time-based user interests (Lim et al. 2015b).
- Proposing the TOURRECINT algorithm for recommending customized tours with a mandatory POI category based on user interests, i.e., the most frequently visited POI category (Lim 2015).
- Introducing the GROUPTOURREC problem and proposing an approach to cluster tourists with similar interests into groups, recommend tours based on group interest preferences and POI popularity, and assign tour guides to lead these groups (Lim et al. 2016).
- Developing an approach for detecting location-centric communities using SST links, which are traditional friendship links augmented with spatial and temporal information (Lim et al. 2015a).

Using a Flickr dataset of tourist visits to POIs in multiple cities, we compare our proposed algorithms against various baselines using evaluation metrics such as precision, recall, F1-score, user interest scores, POI popularity scores, and others. Experimental results show that our proposed algorithms out-perform their respective baselines in terms of these metrics, across all cities. We refer readers to the respective papers (listed above) for a more detailed discussion on these results.

As part of future work, we also described our plans to adopt a game theoretic approach to tour recommendation. For this work, we aim to model POI "crowdedness" as a common utility and implement equilibrium strategies to recommend tours that minimize over-crowding at POIs for the tourist population as a whole.

5 Acknowledgments

National ICT Australia (NICTA) is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program. The author thanks Shanika Karunasekera, Christopher Leckie and Jeffrey Chan for their useful comments and discussions.

References

Amer-Yahia, S.; Roy, S. B.; Chawlat, A.; Das, G.; and Yu, C. 2009. Group recommendation: Semantics and efficiency. In *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB'09)*, 754–765.

Ardissono, L.; Goy, A.; Petrone, G.; Segnan, M.; and Torasso, P. 2003. Intrigue: personalized recommendation of tourist attractions for desktop and hand held devices. *Applied Artificial Intelligence* 17(8-9):687–714.

Berkelaar, M.; Eikland, K.; and Notebaert, P. 2004. lpsolve: Open source (mixed-integer) linear programming system. http://lpsolve.sourceforge.net/.

Blondel, V. D.; Guillaume, J.-L.; Lambiotte, R.; and Lefebvre, E. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008(10):P10008.

Boratto, L., and Carta, S. 2011. State-of-the-art in group recommendation and new approaches for automatic identification of groups. In *Information Retrieval and Mining in Distributed Environments*, 1–20. Springer Berlin Heidelberg.

Brilhante, I.; Macedo, J. A.; Nardini, F. M.; Perego, R.; and Renso, C. 2013. Where shall we go today? Planning touristic tours with TripBuilder. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management (CIKM'13)*, 757–762.

Brilhante, I.; Macedo, J. A.; Nardini, F. M.; Perego, R.; and Renso, C. 2014. TripBuilder: A tool for recommending sightseeing tours. In *Proceedings of the 36th European Conference on Information Retrieval (ECIR'14)*, 771–774.

Brilhante, I. R.; Macedo, J. A.; Nardini, F. M.; Perego, R.; and Renso, C. 2015. On planning sightseeing tours

with TripBuilder. *Information Processing & Management* 51(2):1–15.

Castillo, L.; Armengol, E.; Onaindía, E.; Sebastiá, L.; González-Boticario, J.; Rodríguez, A.; Fernández, S.; Arias, J. D.; and Borrajo, D. 2008. SAMAP: An user-oriented adaptive system for planning tourist visits. *Expert Systems with Applications* 34(2):1318–1332.

Chen, C.; Zhang, D.; Guo, B.; Ma, X.; Pan, G.; and Wu, Z. 2015. TripPlanner: Personalized trip planning leveraging heterogeneous crowdsourced digital footprints. *IEEE Transactions on Intelligent Transportation Systems* 16(3):1259–1273.

Choudhury, M. D.; Feldman, M.; Amer-Yahia, S.; Golbandi, N.; Lempel, R.; and Yu, C. 2010. Automatic construction of travel itineraries using social breadcrumbs. In *Proceedings of the 21st ACM Conference on Hypertext and Hypermedia (HT'10)*, 35–44.

Cohen, R., and Katzir, L. 2008. The generalized maximum coverage problem. *Information Processing Letters* 108(1):15–22.

Garcia, I.; Sebastia, L.; Onaindia, E.; and Guzman, C. 2009. A group recommender system for tourist activities. In *Proceedings of the 10th International Conference on E-Commerce and Web Technologies (EC-WEB'09)*, 26–37.

Garcia, I.; Sebastia, L.; and Onaindia, E. 2011. On the design of individual and group recommender systems for tourism. *Expert Systems with Applications* 38(6):7683–7692.

Gavalas, D.; Konstantopoulos, C.; Mastakas, K.; and Pantziou, G. 2014. A survey on algorithmic approaches for solving tourist trip design problems. *Journal of Heuristics* 20(3):291–328.

Gionis, A.; Lappas, T.; Pelechrinis, K.; and Terzi, E. 2014. Customized tour recommendations in urban areas. In *Proceedings of the 7th ACM International Conference on Web search and Data Mining (WSDM'14)*, 313–322.

Hu, L.; Cao, J.; Xu, G.; Cao, L.; Gu, Z.; and Cao, W. 2014. Deep modeling of group preferences for group-based recommendation. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI'14)*, 1861–1867.

Jameson, A.; Baldes, S.; and Kleinbauer, T. 2003. Enhancing mutual awareness in group recommender systems. In *Proceedings of the IJCAI Workshop on Intelligent Techniques for Web Personalization*.

Kurashima, T.; Iwata, T.; Irie, G.; and Fujimura, K. 2010. Travel route recommendation using geotags in photo sharing sites. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM'10)*, 579–588.

Kurashima, T.; Iwata, T.; Irie, G.; and Fujimura, K. 2013. Travel route recommendation using geotagged photos. *Knowledge and Information Systems* 37(1):37–60.

Lim, K. H., and Datta, A. 2016. An interaction-based approach to detecting highly interactive twitter communities using tweeting links. In *Web Intelligence*, volume 14, number 1. IOS Press.

Lim, K. H.; Chan, J.; Leckie, C.; and Karunasekera, S. 2015a. Detecting location-centric communities using social-spatial links with temporal constraints. In *Proceedings of the 37th European Conference on Information Retrieval (ECIR'15)*, 489–494.

Lim, K. H.; Chan, J.; Leckie, C.; and Karunasekera, S. 2015b. Personalized tour recommendation based on user interests and points of interest visit durations. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI'15)*, 1778–1784.

Lim, K. H.; Chan, J.; Leckie, C.; and Karunasekera, S. 2016. Towards next generation touring: Personalized group tours. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS'16)*.

Lim, K. H. 2015. Recommending tours and places-of-interest based on user interests from geo-tagged photos. In *Proceedings of the 2015 SIGMOD PhD Symposium (SIGMOD'15)*, 33–38.

Miller, C. E.; Tucker, A. W.; and Zemlin, R. A. 1960. Integer programming formulation of traveling salesman problems. *Journal of the ACM* 7(4):326–329.

Raghavan, U. N.; Albert, R.; and Kumara, S. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E* 76(3):036106.

Refanidis, I.; Emmanouilidis, C.; Sakellariou, I.; Alexiadis, A.; Koutsiamanis, R.-A.; Agnantis, K.; Tasidou, A.; Kokkoras, F.; and Efraimidis, P. S. 2014. myVisitPlanner GR: Personalized itinerary planning system for tourism. In *Proceedings of the 8th Hellenic Conference on Artificial Intelligence (SETN'14)*, 615–629.

Rosvall, M., and Bergstrom, C. T. 2008. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Science* 105(4):1118–1123.

Sebastia, L.; Garcia, I.; Onaindia, E.; and Guzman, C. 2009. e-Tourism: a tourist recommendation and planning application. *International Journal on Artificial Intelligence Tools* 18(5):717–738.

Sinnott, R. W. 1984. Virtues of the Haversine. *Sky and Telescope* 68(158).

Thomee, B.; Shamma, D. A.; Friedland, G.; Elizalde, B.; Ni, K.; Poland, D.; Borth, D.; and Li, L.-J. 2016. YFCC100M: The new data in multimedia research. *Communications of the ACM* 59(2):64–73.

Tsiligirides, T. 1984. Heuristic methods applied to orienteering. *J. of the Operational Research Society* 35(9):797–809.

UNWTO. 2015. United Nations World Tourism Organization (UNWTO) annual report 2014. http://www2.unwto.org/annual-reports.

Vansteenwegen, P.; Souffriau, W.; and Oudheusden, D. V. 2011. The orienteering problem: A survey. *European Journal of Operational Research* 209(1):1–10.

Yahoo! Webscope. 2014. Yahoo! Flickr Creative Commons 100M dataset (YFCC-100M). http://webscope.sandbox.yahoo.com/catalog.php?datatype=i&did=67.

Constructing Plan Trees for Simulated Penetration Testing

Dorin Shmaryahu
Information Systems Engineering
Ben Gurion University
Israel

Abstract

Penetration Testing (pentesting), where network administrators automatically attack their own network to identify and fix their vulnerabilities, has recently received attention from the AI community. Smart algorithms that can identify robust and efficient attack plans can imitate human hackers better than simple protocols. Current classical planning methods for pentesting model poorly the real world, where the attacker has only partial information concerning the network. On the other hand POMDP-based approaches provide a strong model, but fail to scale up to reasonable model sizes. In this paper we offer a more realistic model of the problem, allowing for partial observability and non-deterministic action effects, by modeling pentesting as a partially observable contingent problem. We suggest several optimization criteria, including worst case, best case, and fault tolerance. We experiment with benchmark networks, showing contingent planning to scale up to large networks.

1 Introduction

Penetration testing (pentesting) is a popular technique for identifying vulnerabilities in networks, by launching controlled attacks (Burns et al. 2007). A successful, or even a partially successful attack reveals weaknesses in the network, and allows the network administrators to remedy these weaknesses. Such attacks typically begin at one entrance point, and advance from one machine to another, through the network connections. For each attacked machine a series of known exploits is attempted, based on the machine configuration, until a successful exploit occurs. Then, this machine is controlled by the attacker, who can launch new attacks on connected machines. The attack continues until a machine inside the secure network is controlled, at which point the attacker can access data stored inside the secured network, or damage the network.

In automated planning the goal of an agent is to produce a plan to achieve specific goals, typically minimizing some performance metric such as overall cost. There are many variants of single agent automated planning problems, ranging from fully observable, deterministic domains, to partially observable, non-deterministic or stochastic domains. Automated planning was previously suggested as a tool for conducting pentesting, exploring the two extreme cases — a classical planning approach, where all actions are deterministic, and the entire network structure and machine configuration are known, and a POMDP approach, where machine configuration are unknown, but can be noisily sensed, and action outcomes are stochastic.

The classical planning approach scales well for large networks, and has therefore been used in practice for pentesting. However, the simplifying assumptions of complete knowledge and fully deterministic outcomes results in an overly optimistic attacker point-of-view. It may well be that a classical-planning attack has a significantly lower cost than a real attack, identifying vulnerabilities that are unlikely to be found and exploited by actual attackers.

The POMDP approach on the other hand, models the problem better, and can be argued to be a valid representation of the real world. One can model the prior probabilities of various configurations for each machine as a probability distribution over possible states, known as a belief. Pinging actions, designed to reveal configuration properties of machines are modeled as sensing actions, and a probability distribution can be defined for the possible failure in pinging a machine. The success or failure of attempting an exploit over a machine can be modeled as a stochastic effect of actions.

This approach, however, has two major weaknesses — first, POMDP solvers do not scale to the required network size and possible configurations. Second, a POMDP requires accurate probability distributions for initial belief, sensing accuracy, and action outcomes. In pentesting, as in many other applications, it is unclear how the agent can reliably obtain these distributions. In particular, how to identify an accurate probability distribution over the possible OS for the machines in the network? Prior work (Sarraute *et al.*) has devised only a first over-simplifying model of "software updates", which the authors admit themselves is not suitable and may adversely affect the usefulness of the pentesting result ("garbage in, garbage out"). One might consider research into obtaining better distributions, e.g. by statistics from data, but this is wide open, and in any case the scalability weakness remains.

A possible simple approach to defining such probabilities is to use a uniform distribution. However, a solution to a POMDP defined using a uniform distribution can be arbitrar-

ily bad. Consider, for example, a case where there exists a large set of configurations that are easy to penetrate, such as a variety of old, unupdated operating systems. All these configurations may be very rare in the network, yet still exist on some machines, and are hence represented in the model. Assuming a uniform distribution over possible configurations, an attacker may believe that these vulnerable configurations are as frequent as any other configuration, and may hence attempt a long sequence of exploits which will work only for these faulty configurations. In such a case, the performance of the agent measured over the uniform POMDP, may be arbitrarily far from its performance in practice.

As an intermediate model between classical planning and POMDPs, MDP models of pentesting have been suggested (Durkota *et al.* 2015; Hoffmann 2015). These somewhat simplify the issue of obtaining the probabilities, now corresponding to "success statistics" for exploits. Yet even this data is not easy to come by in practice, and scalability may still be problematic given that solving factored MDPs is notoriously hard (a thorough empirical investigation has yet to be conducted).

In this paper we suggest another, different, intermediate model between classical planning and POMDPs. We replace the POMDP definition with partially observable contingent planning, a qualitative model where probability distributions are replaced with sets of possible configurations or action effects (Albore *et al.* 2009; Muise *et al.* 2014; Komarnitsky and Shani 2014). Solvers for this type of models scale better than POMDP solvers, and can be used for more practical networks. As these models require no probabilities, we avoid the guesswork inherent in their specification.

Contingent planners attempt to find a plan tree (or graph), where nodes are labeled by actions, and edges are labeled by observations. This plan tree is a solution to the problem if all leaves represent goal states. In pentesting, one is also interested in finding better attacks, i.e. in ranking the set of possible plan trees by some measurable quantity. For example, an attacker may be interested in attacks that, at the worst case, take no more than a certain amount of time. An important research question is, hence, to define possible optimization criteria for attack plan trees. Then, one must design algorithms dedicated to these optimization criteria.

We focus here on the first question — possible optimization criteria for ranking contingent plan trees. We suggest a number of such criteria, including best and worst case, budget constrained plans, and fault-tolerant planning (Domshlak 2001). We also consider deadends, which arise in pentesting as some machine configurations cannot be penetrated, leaving no opportunity to the attacker to reach its goal. We discuss how to define and compare contingent plans under such unavoidable deadends.

We demonstrate empirically that different heuristics produce different plan trees, and that these plan trees can be compared using our optimization criteria, to prefer on heuristic over another. We leave the construction of optimal and approximate contingent planners for future research.

2 Networks and Pentesting

We begin by providing a short background on pentesting.

We can model networks as directed graphs whose vertices are a set M of *machines*, and edges representing connections between pairs of $m \in M$. Like previous work in the area, we assume below that the attacker knows the structure of the network. But this assumption can be easily removed in our approach. We can add sensing actions that test the outgoing edges from a controlled host to identify its immediate neighbors. From an optimization prespective, though, not knowing anything about the network structure, makes it difficult to create smart attacks, and the attacker is forced to blindly tread into the network. It might well be that some partial information concerning the network structure is known to the attacker, while additional information must be sensed. We leave discussion of interesting forms of partial knowledge to future work.

Each machine in the network can have a different *configuration* representing its hardware, operating system, installed updates and service packs, installed software, and so forth. The network configuration is the set of all machine configurations in the network.

Machine configuration may be revealed using sensing techniques. For example, if a certain series of 4 TCP requests are sent at exact time intervals to a target machine, the responses of the target machine vary between different versions of Windows (Lyon 2009). In many cases several different such methods must be combined to identify the operating system. Sending such seemingly innocent requests to a machine to identify its configuration is known as fingerprinting. Not all the properties of a target machine can be identified. For example, one may determine that a certain machine runs Windows XP, but not which security update is installed.

Many configurations have *vulnerabilities* that can be *exploited* to gain control over the machine, but these vulnerabilities vary between configurations. Thus, to control a machine, one first pings it to identify some configuration properties, and based on these properties attempts several appropriate exploits. As the attacker cannot fully observe the configuration, these exploits may succeed, giving the attacker full control of the target machine, or fail as some undetectable configuration property made this exploit useless.

The objective of penetration testing (pentesting) is to gain control over certain machines that possess critical content in the network. We say that a machine m is *controlled* if it has already been hacked into, and the attacker can use it to fingerprint and attack other machines. A *reached* machine m is connected to a controlled machine. All other machines are *not reached*. We assume that the attacker starts controlling the internet, and all machines that are directly connected to the internet are reached.

We will use the following (small but real-life) situation as an illustrative example (Sarraute *et al.*):

Example 2.1. The attacker has already hacked into a machine m', and now wishes to attack a reached machine m. The attacker may try one of two exploits: *SA*, the "Symantec Rtvscan buffer overflow exploit"; and *CAU*, the "CA Unicenter message queuing exploit". SA targets a particular version of "Symantec Antivirus", that usually listens on port 2967. CAU targets a particular version of "CA Unicen-

ter", that usually listens on port 6668. Both work only if a protection mechanism called *DEP* ("Data Execution Prevention") is disabled. The attacker cannot directly observe whether DEP is enabled or not.

If SA fails, then it is likely that CAU will fail as well because DEP is enabled. Hence, upon observing the result of the SA exploit, the attacker learns whether DEP is enabled. The attacker is then better off trying other exploits else. Achieving such behavior requires the attack plan to observe the outcomes of actions, and to react accordingly. Classical planning which assumes perfect world knowledge at planning time cannot model such behaviors.

3 Contingent Planning Model and Language

A contingent planning problem is a tuple $< P, A_{act}, A_{sense}, \phi_I, G >$, where P is a set of propositions, A_{act} is a set of actuation actions, and A_{sense} is a set of sensing actions. An actuation action is defined by a set of preconditions — propositions that must hold prior to executing the actions, and effects — propositions that hold after executing the action. Sensing actions also have preconditions, but instead of effects they reveal the value of a set of propositions. ϕ_I is a propositional formula describing the set of initially possible states. $G \subset P$ is a set of goal propositions.

In our pentesting application, P contains propositions for describing machine configuration, such as $OS(m_i, winxp)$, denoting that machine m_i runs the OS Windows XP. Similarly, $SW(m_i, IIS)$ represents the existence of the software IIS on machine m_i. In addition, the proposition $controlling(m_i)$ denotes that the attacker currently controls m_i, and the proposition $hacl(m_i, m_j, p)$ denotes that machine m_i is directly connected to machine m_j through port p.

The set A_{sense} in our pentesting model represents the set of possible queries that one machine can launch on another, directly connected machine, pinging it for various properties, such as its OS, software that runs on it, and so forth. Each such sensing action requires as precondition only that the machines will be connected, and reveals the value of a specific property. In some cases there are certain "groups" of operating systems, such as Windows XP with varying service packs and updates installed. In this case we can allow one property for the group ($OS(m_i, winxp)$) and another property for the version, such as ($OSVersion(m_i, winxp_sp1)$) which may not be observable by the attacker.

The set A_{act} in our pentesting model contains all the possible exploits. We create an action $a_{e,m_{source},m_{target}}$ for each exploit e and a pair of directly connected machines m_{source}, m_{target}. If an exploit e is applicable only to machines running Windows XP, then $OS(m_{target}, winxp)$ would appear in the preconditions. Another precondition is $controlling(m_{source})$ denoting that the attacker must control m_{source} before launching attacks from it. The effect of the action can be $controlling(m_{target})$, but we further allow the effect to depend on some hidden property p that cannot be sensed. This is modeled by a conditional effect $\langle p, controlling(m_{target})\rangle$ denoting that if property p exists on m_{target} than following the action the attacker controls m_{target}.

Belief states in contingent planning are sets of possible states, and can often be compactly represented by logic formulas. The initial belief formula ϕ_I represents the knowledge of the attacker over the possible configurations of each machine. For example $oneof(OS(m_i, winxp), OS(m_i, winnt4), OS(m_i, win7))$ states that the possible operating systems for machine m_i are Windows XP, Windows NT4, and Windows 7.

Like Sarraute et al., we assume no non-determinism, i.e., if all properties of a configuration are known, then we can predict deterministically whether an exploit will succeed. We do allow for non-observable properties, such as the service pack installed for the specific operating system. We support actions for sensing whether an exploit has succeeded. Hence, observing the result of an exploit action reveals information concerning these hidden properties.

Example 3.1. We illustrate the above ideas using a very small example, written in a PDDL-like language for describing contingent problems (Albore *et al.* 2009).

We use propositions to describe the various properties of the machines and the network. For example, the predicate *(hacl* $?m_1$ $?m_2$) specifies whether machine m_1 is connected to machine m_2, and the predicate *(HostOS ?m ?o)* specifies whether machine m runs OS o. While in this simple example we observe the specific OS, we could separate OS type and edition (say, Windows NT4 is the type, while Server or Enterprise is the edition). We can then allow different sensing actions for type and edition, or allow only sensing of type while edition cannot be directly sensed.

We define actions for pinging certain properties. For example, the *ping-os* action:

```
(:action ping-os
    :parameters (?s - host ?t - host ?o - os)
    :precondition (and (hacl ?s ?t)
                       (controlling ?src)
                       (not(controlling ?target))
    :observe (HostOS ?target ?o)
)
```

allows an attacker that controls host s connected to an uncontrolled host t, to ping it to identify whether it's OS is o. We allow for a similar ping action for installed software.

The *exploit* action attempts to attack a machine exploiting a specific vulnerability:

```
(:action exploit
    :parameters (?s - host ?t - host ?o - os ?sw - sw ?v - vuln)
    :precondition (and (hacl ?s ?t)
                       (controlling ?s)
                       (not(controlling ?t))
                       (HostOS ?t ?o)
                       (HostSW ?t ?s)
                       (Match ?o ?sw ?v))
    :effect (when (ExistVuln ?v ?t) (controlling ?t))
)
```

The preconditions specify that the machines must be connected, that the OS is o and the software sw is installed, and that the vulnerability v which we intend to exploit matches the specific OS and software.

The success of the exploit depends on whether the vulnerability exists on the target machine, which manifests in the conditional effect. The attacker cannot directly observe whether a specific vulnerability exists, but can use the *CheckControl* action to check whether the exploit has succeeded:

```
(:action CheckControl
    :parameters (?src - host ?target - host)
    :precondition (and (hacl ?src ?target ?p) (controlling ?src))
    :observe (controlling ?target)
)
```

The initial state of the problem describes the knowledge of the attacker prior to launching an attack:

```
(:init
1:    (controlling internet)
2:    (hacl internet host0)
      (hacl internet host1)
      (hacl host1 host2)
      (hacl host0 host2)
      ...
3:    (oneof (HostOS host0 winNT4ser) (HostOS host0 winNT4ent))
      (oneof (HostOS host1 win7ent) (HostOS host1 winNT4ent))
      ...
4:    (oneof (HostSW host0 IIS4) (HostSW host1 IIS4))
      ...
5:    (Match winNT4ser IIS4 vuln1)
      ...
6:    (or (ExistVuln vuln1 host0) (ExistVuln vuln2 host0))
      ...
)
```

We state that initially the attacker controls the "internet" only (part 1). In this case the structure of the network is known, described by the *hacl* statements (part 2). Then, we describe which operating systems are possible for each of the hosts (part 3). Below, we specify that either $host0$ or $host1$ are running the software IIS (part 4). We describe which vulnerability is relevant to a certain OS-software pair (part 5), and then describe which vulnerabilities exit on the various hosts (part 6).

The above specification may allow for a configuration where no vulnerability exists on a host (machine) that matches the host OS and software. Hence, none of the exploits will work for that specific host.

4 Plan Trees and Optimization Criteria

We now formally define solutions to a contingent planning problem. We discuss deadends that arise in pentesting, and then turn our attention to a discussion of optimization criteria.

4.1 Contingent Plan Trees

A solution to a contingent planning problem is a plan tree, where nodes are labeled by actions. A node labeled by an actuation action will have only a single child, and a node labeled by an observation action will have multiple children, and each outgoing edge to a child will be labeled by a possible observation.

An action a is applicable in belief state b, if for all $s \in b$, $s \models pre(a)$. The belief state b' resulting from the execution of a in b is denoted $a(b)$. We denote the execution of a sequence of actions $a_1^n =< a_1, a_2, ..., a_n >$ starting from belief state b by $a_1^n(b)$. Such an execution is valid if for all i, a_i is applicable in $a_1^{i-1}(b)$.

Plan trees can often be represented more compactly as plan graphs(Komarnitsky and Shani 2014; Muise *et al.* 2014), where certain branches are unified. This can lead to a much more compact representation, and to scaling up to larger domains. Still, for ease of exposition, we discuss below plan trees rather than graphs.

In general contingent planning, a plan tree is a solution, if every branch in the tree from the root to a leaf, labeled by actions a_1^n, $a_1^n(b_I) \models G$. In pentesting, however, it may not be possible to reach the goal in all cases, because there may be network configurations from which the target machine simply cannot be reached. To cater for this, we need to permit plan trees that contain *dead-ends*. We define a dead-end to be a state from which there is no path to the goal, given *any* future sequence of observations. That is, any plan tree starting from a dead-end state would not reach the goal in any of its branches. For example, a dead-end state arises if no exploit is applicable for the goal machine. It is clearly advisable to stop the plan (the attack) at such states. On the other hand, if a state is not a dead-end, then there still is a chance to reach the target so the plan/attack should continue.

There is hence need to define contingent plans where some of the branches may end in dead-ends. A simple solution, customary in probabilistic models, is to introduce a give-up action which allows to achieve the goal from any state. Setting the cost of that action (its negative reward) controls the extent to which the attacker will be persistent, through the mechanism of expected cost/expected reward.

In a qualitative model like ours, it is not as clear what the cost of giving up (effectively, of flagging a state as "dead-end" and disregarding it) should be. It may be possible to set this cost high enough to force the plan to give up only on dead-ends as defined above. But then, the contingent planner would effectively need to search all contingent plans not giving up, before being able to give up even once.

We therefore employ here a different approach, allowing the planner to give-up on s iff it can prove that s is a dead-end. Such proofs can be lead by classical-planning dead-end detection methods, like relaxation/abstraction heuristics, adapted to our context by determinizing the sensing actions, allowing the dead-end detector to choose the outcome. In other words, we employ a sufficient criterion to detect dead-end states, and we make the give-up action applicable only on such states. As, beneath all dead-ends, eventually the pentest will run out of applicable actions, eventually every dead-end will be detected and the give-up enabled.

In general, this definition would not be enough because the planner could willfully choose to move into a dead-end, thereby "solving" the task by earning the right to give up. This cannot happen, however, in the pentesting application, as all dead-ends are *unavoidable*, in the following sense. Say N is a node in our plan tree T, and denote by $[N]$ those initial states from which the execution of T will reach N. If N is a dead-end, then every $I \in [N]$ is unsolvable, i.e., there does not exist any sequence of A_{act} actions leading from I to the goal. In other words, any dead-end the contingent plan

may encounter is, in the pentesting application, inherent in the initial state. Matters change if we impose a budget limit on the attack, in which case the dead-ends encountered depend on which decisions are taken. We define an according plan quality criterion as part of the next subsection.

4.2 Optimization Criteria for Contingent Plans

General contingent planning follows the satisfying planning criterion, that is, one seeks any solution plan tree. It is possible, though, to consider cases where one plan tree is preferable to another, and construct algorithms that seek better, or even the best possible plan tree.

When we assume that the environment is modeled as a POMDP, and we know all the probability distributions, an obvious optimization criterion is the expected discounted reward (or cost) from executing a plan tree in the environment, and can be estimated by running multiple trials and computing the average discounted reward (ADR). In this paper, however, we focus on cases where these distributions are unknown. Without the specified distributions one cannot accurately estimate expected reward. Any attempt to use a different distribution, such as a uniform distribution, which may be arbitrarily far from the true distribution, may result in quality estimation that is arbitrarily far from reality.

We hence revert to other possible optimization criteria. Perhaps the most trivial optimization criteria under unknown probability distributions is the best case scenario, or the worst case scenario. In the best case scenario we compare plan trees based on the length of the shortest branch leading to a goal state. In the worst case scenario we compare the length of the longest branch leading to a goal state, preferring plan trees with shorter worst case branches. This may be somewhat different than the naive definition of a worst case, as a complete failure is obviously worse (less desirable) than a success after a lengthy sequence of actions. In our case, as the deadends in the plan trees are unavoidable, the naive worst case — a complete failure — is identical in all plan trees. We thus choose to ignore branches ending with deadends when considering worst case analysis.

While well defined, best and worst case optimization may not be sufficiently expressive. A best case scenario is too optimistic, assuming that all attack attempts will be successful. A worst case scenario is over pessimistic, assuming that all attack attempts, but the last one, will fail. We would like to define finer optimization criteria.

Budget Optimization One possible such criterion assumes attacks on a budget — that is, the attacker is allowed only a certain predefined number of actions (or total cost) in a branch. When the budget runs out, the attacker is not allowed any additional actions, and hence, a deadend occurs. Setting a budget prior to attacking seems like a reasonable requirement from an attacker. For example, if action costs represent the time it takes for each action, the attacker may wish to restrict attention only to attacks that require less than a certain amount of time.

Now, given two plan trees that respect a given budget, we can compare them on two possible criteria — the best case scenario and the set of solved network configurations. The worst case scenario is less interesting here as it will probably be identical to the budget.

The set of network configurations where the attacker has reached the goal under the budget is now interesting, because deadends induced by the budget may well be avoidable. That is, one can choose different attack plans, that may lead to the goal faster and hence will result in less deadends. However, simply counting the number of network configurations for which the goal has been reached is undesirable under our qualitative assumptions. For example, it may well be that plan tree τ_1 solves only for a single configuration c, while another plan tree τ_2 solves for all configurations but c. Still, it may be that the (unknown) probability of c is 0.9, making τ_1 preferable to τ_2. As we do not know these probabilities, we cannot make such comparisons.

We can hence only declare plan tree τ_1 to be better than plan tree τ_2 if the set of solved configurations ofτ_1 is a strict superset of the set of solved configurations of τ_2. As contingent planners typically maintain some type of belief over the set of possible network configurations in each search node, such computations are feasible. For example, if the belief is maintained by a logic formula, as we do, then each goal leaf g has a logic formula ϕ_g defining the belief at that leaf. We can check whether

$$\bigvee_{g \in G(\tau_1)} \phi_g \models \bigvee_{g \in G(\tau_2)} \phi_g \quad (1)$$

$$\bigvee_{g \in G(\tau_2)} \phi_g \not\models \bigvee_{g \in G(\tau_1)} \phi_g \quad (2)$$

where $G(\tau)$ is the set of goal leaves in plan tree τ.

Fault Tolerance Optimization Another possible optimization is by extending the ideas of fault-tolerance planning to pentesting. In fault-tolerance planning (Domshlak 2001), assuming that certain actions may fail with some low probability, a solution achieves the goal under the assumption that no more than k failures will occur. The underlying assumption is that the probability of more than k failures is so small, that we can ignore it. A failure in our case can be defined in one of two ways — either that we will ping a machine for a given property (say, $OS(m_i, winxp)$) and receive a negative response. Alternatively, we may declare a failure only when we attempt an exploit, and it fails to achieve control of a machine (due to some unobserved property).

With that view in mind, we can compare solution plan trees, focusing only on branches that contain exactly k failures. As having no more than k failures is an optimistic assumption, it is reasonable to check the worst case under this optimistic assumption. That is, of the branches of the plan tree that have the lowest probability that we care about, we compare the longest branches. Looking at the best case — the shortest branch when having no more than k failures, is identical to the overall best case scenario, ignoring failures all together.

A complementing approach assumes no less than k failures at each branch. This assumption is more appropriate where the probability of failure is sufficiently large, such that the probability of completing a task without any failure

is very low. In such cases, we again compare only branches with exactly k branches, and as no less than k failures is a pessimistic assumption, we compare the best case scenario — the shortest branch with exactly k failures. Again, the worst case is less interesting as it is identical to the overall worst case.

5 Research plan

We now discuss the next steps on our research agenda. We have four different designed items:

1. Improving the configuration of the network.
2. Getting real network data.
3. Finding additional heuristics.
4. Empirical validation.

5.1 Improving the Configuration of the Network

Currently we believe that we provide a stronger and more realistic model of the problem than previous approaches, that can scale up to reasonably sized networks. However, we consider changing some of the possible configuration, making the model even more realistic.

We intend to add OS families — generalizing the OS configuration under the assumption that we can not sense a specific Os, but only a general category of an Os. As we explain above, the sensing procedure for machine configuration is known as fingerprinting. In the real world this process may identify the OS family and not the specific one. For example, if a certain machine runs Windows XP SP3, the observation about the target machine OS may be Windows XP.

We can support hidden connections between machines (sensing for connections). That way we eliminate the need for planner knowledge over the hosts connection. We have doubts about hiding those connection. Making the planner work without previous knowledge will make all states and action to seem equally valuable — choosing to attack one machine is equivalent to choosing to attack another. For example, our current heuristics choose to attack a machine that is closest to the target machine, which cannot be done when network connections are unknown.

5.2 Getting real network data

We are working on getting some real data network configuration using Nmap (Network Mapper) tool to gathering all configuration data (OS, Softwares, connections and vulnerabilities) from existing network. The Nmap tool is a free and open source utility for network discovery and security auditing. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what applications those hosts are offering, what operating systems they are running. First we need to run Nmap on a small network showing there is no risk to execute it in a larger network, and then hopefully we could run Nmap on the university network.

5.3 Identifying Useful Heuristics

Our planning algorithm is a best first contingent planner. We use a heuristic to determine which state and action to expand next.

We are interested in generating a variety of plan trees, given different heuristics. Currently, our planner supports 4 different heuristics. The first two heuristics, random and LIFO are not useful as they cannot create attack graphs for large networks. We will use them only as a baseline to show that the other heuristics are better given the optimization criteria. Our main goal is to develop new heuristics that will be scalable to larger networks and will produce better plan trees.

Random Heuristic This heuristic gives each state a random heuristic value. It preform badly and is not scalable to larger networks of more than 5 hosts. This heuristic will be our baseline for comparing plans trees in small networks.

LIFO Heuristic A LIFO heuristic gives the highest value to the last state. The last state to arrive is the next state the planner will expand. This heuristic also preforms badly and is not scalable to larger networks of more than 5 hosts.

Shortest Path Heuristic This heuristic chooses the next host to attack among the reachable hosts closest to the goal host. Attack actions for the chosen host are selected at random. This heuristic is more scalable than the previous two, handling networks with up to 16 hosts.

Shortest Path and Action Selection Heuristic We chose the next host to attack from the hosts closest to the goal host. When choosing actions, we first ping a host for its operating system, and then we ping it only for software that, combined with the observed OS, may have a vulnerability. If a possible vulnerability has been detected, we attempt an exploit, followed by a sensing action to check if control was gained over the attacked host. This simple heuristic, proves to be highly effective for this application, and we manage to produce attack graphs for networks with 80 hosts and more.

5.4 Empirical Study

We now review the experiments that we want to conduct in order to show the contribution of using contingent planning for pentesting .We want to demonstrate that the criteria we suggest can be used to differentiate between various plan trees (graphs), helping us to select a better algorithm. First, we will generate a number of networks of varying sizes using the generator of Hoffman and Steinmetz(Sarraute *et al.* ; Steinmetz *et al.*).

We experiment with a simple greedy best first contingent planner that uses a heuristic to determine which state and action to expand next. In addition, we use a mechanism for detecting repeated plan tree nodes, converting the plan tree into a plan graph. We augment this algorithm with a domain specific deadend detection mechanism, checking whether there is still a path from the attack source ("the internet") to the target host.

We will employ several domain specific heuristics (as we explained above), that leverage the network graph. Generat-

ing several different plan graphs. We can compare the runtime and scalabilty of the various heuristics.

We will run the heuristics over various network sizes. This allows us to employ the optimal solution criteria to conclude which plan is better. We can calculate the best and worst case in the fault tolerance scenario, running the planner with different values of k and compare the best (shortest) and worst (longest) path to goal. We will say that a plan graph 'A' is better than another plan graph 'B' if in a given k failures the longest path to the goal in 'A' is shorter than the longest path to goal in 'B'.

We also need to compare between the sets of initial states where the goal can be reached in each plan graph. Let $s(A)$ be the set of states for which the goal can be reached in plan graph A and $s(B)$ to be the set of such states in plan graph B. If $s(B) \subset s(A)$ we can say that plan graph A is better than plan graph B. Identifying the set of initial states for which there is a solution, and comparing sets is not a trivial problems, and we must identify efficient methods for doing that.

References

Alexandre Albore, Héctor Palacios, and Hector Geffner. A translation-based approach to contingent planning. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1623–1628, 2009.

Burns et al. *Security Power Tools*. O'Reilly Media, 2007.

Carmel Domshlak. Fault tolerant planning: Complexity and compilation. volume 22, pages –, 2001.

Karel Durkota, Viliam Lisý, Branislav Bosanský, and Christopher Kiekintveld. Optimal network security hardening using attack graph games. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 526–532, 2015.

Jörg Hoffmann. Simulated penetration testing: From "dijkstra" to "turing test++". In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, pages 364–372, 2015.

Radimir Komarnitsky and Guy Shani. Computing contingent plans using online replanning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2322–2329, 2014.

Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

Christian J. Muise, Vaishak Belle, and Sheila A. McIlraith. Computing contingent plans via fully observable non-deterministic planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 - 31, 2014, Québec City, Québec, Canada.*, pages 2322–2329, 2014.

Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. POMDPs make better hackers: Accounting for uncertainty in penetration testing.

Marcel Steinmetz, Jörg Hoffmann, and Olivier Buffet. Revisiting goal probability analysis in probabilistic planning.

Optimization Approaches to Multi-robot Planning and Scheduling

Kyle E. C. Booth
Department of Mechanical & Industrial Engineering
University of Toronto, Toronto, Ontario, Canada
kbooth@mie.utoronto.ca

The study and use of multi-robot teams has become more prevalent within academia and industry as the capability and autonomy of these systems continues to improve (Arai, Pagello, and Parker 2002). With high levels of progress already made concerning the control of individual robots, the acknowledged advantages of multi-robot systems (MRS) has resulted in considerable research attention in the last couple of decades in efforts to build more efficient systems for their coordination (Gerkey and Matarić 2004). These advantages include resolving more complex tasks, increasing the speed at which tasks can be completed, and enhancing the level of system reliability and redundancy present within single-robot solutions (Yan, Jouandeau, and Cherif 2013).

This thesis, though in early stages of development, concerns the field of study within MRS known as *multi-robot task allocation* (MRTA) (Gerkey and Matarić 2004). Specifically, it proposes to investigate the integration of techniques from the optimization literature, namely mixed-integer and constraint programming, within architectures for MRTA. This research area aims to solve multi-robot coordination problems pertaining to task distribution to robot resources and the temporal scheduling of tasks on such resources. These problems have a wide variety of real-world applications including planetary exploration (Mataric and Sukhatme 2001), airport and harbor transhipment (Alami et al. 1998), and emergency response (Østergård, Matarić, and Sukhatme 2001).

Multi-robot Task Allocation

Given a team of cooperating robots, a set of tasks that need to be completed, and a problem-specific cost function, the most fundamental instance of MRTA involves determining a mapping of tasks to robots such that the cost function is minimized. Indeed, when the number of robots and tasks are equal and the mapping is one-to-one, the problem can be represented by the classical *linear assignment problem*, solvable in $O(n^3)$ time with a modificiation of the Hungarian method (Kuhn 1955). However, MRTA problems often contain more complex objective functions or the need for tasks to be allocated *and* scheduled on available resources. In such cases, the underlying problem frequently becomes provably $\mathcal{NP}$-Hard, taking on the form of other classical problems within the combinatorial optimization literature such as the Multiple Traveling Salesman Problem (m-TSP) (Papadimitriou 1977). Moreover, the presence of complex constraints (e.g. precedence relationships) between tasks within and across robot schedules further contributes to the difficulty of solving such problems. These *time-extended* allocation problems with *task-dependencies* (Korsah, Stentz, and Dias 2013) are often approached with sophisticated heuristic techniques that sacrifice solution optimality in favour of faster convergence.

Existing Approaches

Early efforts to develop solutions for MRTA include iterated assignment architectures such as ALLIANCE (Parker 1998) and M+ (Botelho and Alami 1999). These architectures employ dispatch-style algorithms where single tasks are assigned and executed before subsequent allocations are made. Various decentralized and fully-distributed techniques have also been proposed for these problems, notably the market-based (Dias et al. 2006) and auction-based (Gerkey and Matari 2002) methods developed within the robotics community. More recently, there have been efforts to use linear and mixed-integer programming (MIP) techniques from the operations research (OR) community to solve MRTA problems (Korsah et al. 2012), largely due to attractive bounds on solution quality, though these methods have not been fully exploited as of yet. Constraint programming (CP) has been proposed as a suitable candidate approach for these problems (Van Hentenryck and Saraswat 1996), however, the application of CP to multi-robot task planning and scheduling has been, to the best of our knowledge, limited.

Research Focus

The first component of our ongoing research focuses on the development and application of optimization-based methods to solve single and multi-robot task planning problems. We investigate the use of MIP and CP techniques to produce high-quality robot task plans that yield provable bounds on solution quality. The suitability of these 'model-and-solve' techniques is computationally assessed, and methods for improving algorithm performance through specialized modeling (e.g., symmetry breaking constraints, auxiliary variables) and search manipulation (e.g., branching rules, variable and value ordering heuristics) are implemented. As part of our future research we plan to develop more specialized

algorithms based on hybrid approaches, incorporating concepts from the OR, CP, and AI communities to further enhance performance. We also aim to study the use of our optimization-based methods for plan repair and replanning, algorithm extensions that look to address real-world uncertainty associated with MRTA applications.

The second component of our research explores the integration of these optimization-based techniques within real-world robot architectures, combining environmental perception, achieved through on-board sensors, with our task planning system to achieve truly autonomous decision-making. Within this integration, the task planning methods developed contribute as a high-level mission planner for the system, allocating tasks to each robot team member as well as specifying when and where such tasks should be executed. These integration efforts are realized using the open source Robot Operating System (ROS) (Quigley et al. 2009), which enables the effective management of communication among individual robot subsystems or between multiple robots. In this architecture, we use peer-to-peer communication graph renderings where nodes represent individual subsystems or robots and arcs represent connections between them. Using ROS-specific implementation details such as *messages* and *topics*, we are able to effectively facilitate system communication. Communication is used to coordinate individual robot function, inform the system of changes within the environment, as well as deliver task allocations and commands to the various robot resources. We validate the utility of our methods using both simulated environments and testing on physical robots. For simulated experimentation, we use ROS Visualization software to model the robots and high-level task planner. These experiments are primarily aimed at improving the performance of our task planning methods. Physical robot implementation validates the real-world function of our task planner as well as important lower-level systems including autonomous navigation, path-planning, and object detection. For these tests we use OpenSlam's (openslam.org) GMapping to create our environment map via simultaneous localization and mapping (SLAM).

Research efforts thus far have shown promise through our work on a single-robot task scheduling problem involving a socially-assistive robot facilitating human-robot interactions (HRI) within a retirement home (Booth et al. 2016). The problem involves reasoning about disjoint time windows, robot travel times, intra-schedule task dependencies, and robot energy levels. For the problem studied, CP has been shown to be the dominant technology, finding high-quality solutions in much shorter runtimes than both MIP and temporal planning techniques. This optimization-based approach is integrated on the social robot *Tangy*, using the ROS architecture as previously described. The integration is tested experimentally on a number of realistic scenarios, demonstrating its physical utility as a viable robot task planning alternative. As a natural progression to our work, research efforts are underway to extend this work to multi-robot variants of the problem.

Conclusion

We explore the use of optimization-based methods for multi-robot task allocation (MRTA) problems. Our contributions and ongoing research consist of two components: i) algorithmic development, and ii) system integration and testing.

For the first component, we investigate the use of 'off-the-shelf' applications of mixed-integer programming (MIP) and constraint programming (CP), as well as look into enhancing these approaches through specialized modeling, search manipulation, and hybridization. In the second component, we work towards integrating our planning and scheduling algorithms within a functioning robot architecture, using the Robot Operating System (ROS) to facilitate system communication. We test our methods using simulated environments, achieved with ROS Vizulization software, and validate their utility with implementation on physical robot systems. In future research, we plan to explore algorithmic extensions that incorporate replanning events and plan repair in efforts to address the uncertainty associated with real-world problems.

References

Alami, R.; Fleury, S.; Herrb, M.; Ingrand, F.; and Robert, F. 1998. Multi-robot cooperation in the martha project. *Robotics & Automation Magazine, IEEE* 5(1):36–47.

Arai, T.; Pagello, E.; and Parker, L. E. 2002. Editorial: Advances in multi-robot systems. *IEEE Transactions on robotics and automation* 18(5):655–661.

Booth, K. E.; Tran, T. T.; Nejat, G.; and Beck, J. C. 2016. Mixed-integer and constraint programming techniques for mobile robot task planning. *Robotics and Automation Letters, IEEE* 1(1):500–507.

Botelho, S. C., and Alami, R. 1999. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, 1234–1239. IEEE.

Dias, M. B.; Zlot, R.; Kalra, N.; and Stentz, A. 2006. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE* 94(7):1257–1270.

Gerkey, B. P., and Matari, M. J. 2002. Sold!: Auction methods for multirobot coordination. *Robotics and Automation, IEEE Transactions on* 18(5):758–768.

Gerkey, B. P., and Matarić, M. J. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research* 23(9):939–954.

Korsah, G. A.; Kannan, B.; Browning, B.; Stentz, A.; and Dias, M. B. 2012. xbots: An approach to generating and executing optimal multi-robot plans with cross-schedule dependencies. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 115–122. IEEE.

Korsah, G. A.; Stentz, A.; and Dias, M. B. 2013. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research* 32(12):1495–1512.

Kuhn, H. W. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly* 2(1-2):83–97.

Mataric, M. J., and Sukhatme, G. S. 2001. Task-allocation and coordination of multiple robots for planetary exploration. In *In Proceedings of the 10th International Conference on Advanced Robotics*. Citeseer.

Østergård, E. H.; Matarić, M. J.; and Sukhatme, G. S. 2001. Distributed multi-robot task allocation for emergency handling. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, 821–826. IEEE.

Papadimitriou, C. H. 1977. The euclidean travelling salesman problem is np-complete. *Theoretical Computer Science* 4(3):237–244.

Parker, L. E. 1998. Alliance: An architecture for fault tolerant multirobot cooperation. *Robotics and Automation, IEEE Transactions on* 14(2):220–240.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 5.

Van Hentenryck, P., and Saraswat, V. 1996. Strategic directions in constraint programming. *ACM Computing Surveys (CSUR)* 28(4):701–726.

Yan, Z.; Jouandeau, N.; and Cherif, A. A. 2013. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems* 10.

Session 6

Knowledge Engineering and Applications

Learning Static Constraints for Domain Modeling from Training Plans

Rabia Jilani
School of Computing and Engineering
University of Huddersfield
United Kingdom

Introduction

AI Planning is a pivotal task that has to be performed by every autonomous system. The automated planning (AP) community has demonstrated a need to uplift planning systems from toy problems to capture more complex domains that closely reflect real life applications (e.g. planning space missions, fire extinction ion management and operation of underwater vehicle) - a way to satisfy the aims of Autonomic systems. Generally, AP techniques require correct description of the planning task. These descriptions include the action model that can be executed in the environment, the state of the objects in the environment and the goal to accomplish.

Domain models encode the knowledge of the domains in terms of actions that can be executed and relevant properties. In centralized approach, this domain could be represented as a knowledge base and automated logical reasoning could be used to determine acts. Specifying operator descriptions by hand for planning domain models is time consuming, error prone and still a challenge for the AI planning community.

The domain model acquisition problem has mainly been tackled by exploiting two approaches. On the one hand, knowledge engineering tools for planning have been introduced over time, for supporting human experts in modelling the knowledge. Two particular examples are itSIMPLE (Vaquero et al. 2007) and GIPO (Simpson, Kitchin, and McCluskey 2007). A review of the state of the art is provided by Shah et al. (Shah et al. 2013). Recently, also crowdsourcing has been exploited for acquiring planning domain models (Zhuo 2015). On the other hand, a number of techniques are currently available for automatic domain model acquisition; they rely on example data for deriving domain models. Significant differences can be found in terms of the quantity and quality of the required inputs.

Our research concerns the area of automated acquisition of full or partial domain model from one or more examples of action sequences within the domain under study. The aim is to enhance the LOCM system and to extend the method of Learning Domain Models for AI Planning Engines via Plan Traces first published in ICAPS 2009 by Cresswell, McCluskey and West (Cresswell, McCluskey, and West 2013a). This method is unique in that it requires no prior knowledge; however it can produce a complete domain model from training data i.e. plan traces. As compared to LOCM, other systems require more input assistance. ARMS (Yang, Wu, and Jiang 2007), for example, is a system that learns the domain model in addition to domain constraints and invariants. It makes use of background information as input e.g. predicate definitions of initial and goal states. Similarly SLAF (Shahaf and Amir 2006) learns action schema but also requires definitions of fluents and a partial observation of intermediate states as input. For a detailed overview, the interested reader is referred to (Jilani et al. 2014).

The main drawback of LOCM is that it does not produce static knowledge, and its model lacks certain expressive features. A key aspect of research presented in this abstract is to enhance the technique with the capacity to generate static knowledge. A test and focus for this research is to make LOCM able to learn static relationships in fully automatic way in addition to dynamic relationships which LOCM already learn. As per our knowledge, no domain learning system has previously been developed with the aim to learn merely from a set of action traces.

Research contributions include (i) a development of new approach to effectively identify static relations for a wide range of problems, by exploiting graph analysis; using a two staged domain enhancement process that first learn missing static facts for action model and then embed those facts in the partial domain model to get working PDDL domain model (ii) Rule extraction from both optimal and suboptimal plan traces (iii) A useful debugging tool for improving existing models, which can indicate hidden static relations helpful for pruning the search space (iv) Combined with LOCM, ASCoL can be a useful tool to produce benchmark domains (v) Identification of basic categories of Static facts and its impact on heuristic learning systems.

Learning Problem Definition

Domain-independent planning systems require that domain constraints and invariants are specified as part of the input domain model. In AI Planning, the generated plan is correct provided the constraints of the world in which the agent is operating are satisfied. Specifying operator descriptions by hand for planning domain models that also require domain specific constraints is time consuming, error prone and still a challenge for the AI planning community.

LOCM

The LOCM systems perform automated generation of the dynamic aspects of a planning domain model, i.e. changes in the state of the world occurring due to action execution, by considering a set of plan traces, only. A plan trace is a sequence of actions that when applied in an initial state, reach the desired goals. No additional knowledge about initial, goal or intermediate states is needed by LOCM. In comparison with other systems, LOCM approach require a minimal amount of information; other systems also require at least partial state information.

LOCM is based on the assumption that the output domain model can be represented in an object-centred representation (Cresswell, McCluskey, and West 2013b). Using an object-centred representation, LOCM outputs a set of parameterized Finite State Machines (FSMs) where each FSM represents the behaviour of each object in the learnt action schema. Such FSMs are then exploited in order to identify pre- and post-conditions of the domain operators. Although LOCM requires no background information, it usually requires many plan traces for synthesizing meaningful domain models. Moreover, LOCM is not able to automatically identify and encode static predicates.

One drawback of the LOCM process is that it can induce only a partial domain model which represents the dynamic aspects of objects and dose not identify and encode static aspects. Static aspects can be seen as relations that appear in the preconditions of operators only, and not in the effects. Therefore, static facts never change in the world, but are essential for modelling the correct action execution. This is problematic since most domains require static predicates to both restrict the number of possible actions and correctly encode real-world constraints. This is the case in well-known benchmark domains like Driverlog, in which static predicates represent the connections of roads; the level of floors in Miconic, or the fixed stacking relationships between specific cards in Freecell. Any missing static relations are manually introduced into the domain models provided by the LOCM systems. LOCM manually declares this information in the following form:

$$Static(connected(L1, L2), Drive(L1, L2)).$$

The above mentioned Prolog code line is added manually to the input training data file to make it a part of the output domain model manually. The fact in the first argument of static is added as a precondition of the action in the second operator argument of static, where shared variable names provide the binding between the action and its precondition.

ASCoL

We enhance the output domain model of the LOCM system to capture static domain constraints from the same set of input training plans as used by LOCM to learn dynamic aspects of the world. In this research, a new framework ASCoL (Automated Static Constraint Learner) has been presented, to make constraint acquisition more efficient, by observing a set of training plan traces. Most systems that learn constraints automatically do so by analysing the operators of the planning world. The ASCoL system discovers static constraints by analysing plan traces for correlations in the data. To do this the algorithm analyses the complete set of input plan traces, based on a predefined set of constraints, and deduces facts from it. It then augment components of the LOCM generated domain with enriched static constraints.

```
sequence_task(1, [unstack(b8, b9),
stack(b8, b10), pick-up(b7), stack(b7,
b8), unstack(b9, b1), put-down(b9),
unstack(b1, b3), stack(b1, b9),
unstack(b3, b2), stack(b3, b6),
pick-up(b5), stack(b5, b3), unstack(b7,
b8), stack(b7, b2), unstack(b8, b10),
stack(b8, b7), pick-up(b10), stack(b10,
b5)], _, _).
```

Figure 1: An example of a blocksworld plan formatted as required by LOCM.

We define the learning problem that ASCoL addresses as follows. Given the knowledge about parameter types (T), operators' dynamic definition and predicates (fluents) in the form of a PDDL representation of a partial domain model induced by LOCM, and a set of plan traces (P), how can we automatically identify the static relation predicates that are needed by operators' preconditions? We base our approach on the assumption that the input plan traces contain tacit knowledge about constraints validation/acquisition. Based on that assumption, we can draw correlations in the data by pattern discovery in the training input only.

Specifically, the input to the static constraints learning algorithm ASCoL is specified as a tuple (P, T), where P is a set of plan traces (goal directed or random walk) and T is a set of types of action arguments in P which ASCoL parses from LOCM output. ASCoL does not require dynamic knowledge of the domain generated by LOCM. ASCoL accepts input plans (plan traces) in the same text-based format supported by LOCM i.e. a training sequence of N actions in order of occurrence, which all have the form:

$$A_i(O_{i1}, ..., O_{ij}) \quad \text{for} \quad i = 1, ..., N$$

Where A is the action name and O is the actionâs object name. Each action (A) in the plan is stated as a name and a list of arguments. In each action (Ai), there are j arguments where each argument is an object (O) of the problem.

Each plan trace is a sequence of actions (Figure 1) in the order of occurrence to satisfy some goal, where each action in the sequence contains the name of the action and objects that are affected by that action execution. Input plan traces do not include any initial, goal or intermediate states or constraints. Static constraints are to be learnt by the system.

The objective is to learn a complete set of static preconditions, and embed them into the correct operators in the LOCM-learnt output to enrich the domain with this required static knowledge. We formally define the correctness of the learnt static knowledge by comparison with benchmark domains from past IPCs.

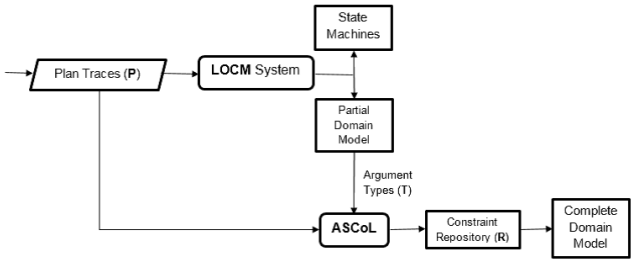


Figure 2: Input Output Structure of ASCoL.

The output for a learning problem is a constraint repository R in PDDL representation that stores all admissible constraints on the arguments of each action A in plan traces P. We assume that input plan traces are noise free while the input domain file at least contains type information for all those operators that the algorithm aims to enhance. Figure 2 shows the general structure of ASCoL in terms of its inputs and outputs.

ASCoL works as a separate unit from LOCM in that LOCM first produces a domain model using a set of plan traces as input. The same LOCM generated domain model, along with the same set of input plan traces, will then be fed to ASCoL to first anticipate the required set of constraints, analyse plan traces and then learn constraints. Finally, it embeds these constraints into the correct operators in the LOCM-learnt output to enrich the domain with this additional static knowledge.

Providing domain constraints to the planning engine may help the planning system in the quick and efficient pruning of the search tree. We aim to capture two major kinds of constraints: domain specific and domain independent constraints. By domain specific we mean the static knowledge that is strictly associated with a domain and is not found as a general example, e.g. the fixed relationships between specific cards in Freecell. Domain independent constraints describe the static knowledge that is generally associated with almost all domains in one way or another; non-equality constraints and link constraints for example.

The ASCoL Methodology

We now briefly present the ASCoL tool that has been developed for identifying useful static relations. The process steps can be summarised as follows:

1. Read the partial domain model (induced by LOCM) and the plan traces.
2. Identify, for all operators, all the pairs of arguments involving the same object types.
3. For each of the pairs, generate a directed graph by considering the objects involved in the matching actions from the plan traces.
4. Analyse the directed graphs for linearity and extract hidden static relations between arguments.
5. Run inequality check.
6. Return the extended domain model that includes the identified static relations.

The main information available for ASCoL comes from the input plan traces. As a first control, we remove from the plan traces all the actions that refer to operators that do not contain at least two arguments of the same type.

Even though, theoretically, static relations can hold between objects of different types, they mostly arise between same-typed objects. This is the case in transport domains, where static relations define connections between locations. Moreover, considering only same-typed object pairs can reduce the computational time required for identifying relations. It is also worth noting that, in most of the cases where static relations involve objects of different types, this is due to a non-optimal modelling process. Furthermore, such relations can be easily identified by naively checking the objects involved in actions; whenever some objects of different type always appear together, they are likely to be statically related.

Types of Static Facts

We describe different kinds of static facts that we came across whilst experimenting with a variety of domains and modelling strategies. We broadly divide static relations into six different types depending upon the structure of knowledge they contain:

1. Locations Map: facts used for representing an underlying map of connected locations. Relations of this kind are mentioned as `adjacent, link, next, path` or `connected` in the domains including TPP, Zenotravel, Storage, Logistics, Mprime, Spanner, Gripper, Trucks and Gold-miner.

2. Level of Specific Object: parameter objects include varying levels of goods, fuel, space and time depending on the scenario of the domain. Domains that mention such static facts include Mprime, Barman, Trucks, TPP and ZenoTravel.

3. Unordered Sequence: ascending or descending sequence of objects but not in any specific flow. The best example is the sequence of floors in the Miconic domain, where a person can board from any floor and can depart on any other floor (up or down).

4. Ordered Sequence: mentioned as `successor, next` and `link` in different domains including specifically Freecell and other card game domains. Here, such static facts allow the sequential arrangement of cards in card stacks among columns, reserve cells and home cells.

5. Compound Relations: static relations –usually exploited in the encoding of card games– that express two independent static relations in terms of one, i.e `can-stack (card1 card2)`. The intuition behind this is the conventional stacking rule based on card suit and rank order. But this can be decomposed into two separate static facts that can then fulfil our criteria of graph analysis, i.e `can-stack-rank(rcard1 rcard2)` and `can-stack-suit(scard1 scard2)`.

6. Non-equality: the best example for these kinds of static facts are Ferry domain and other general transportation domains where two location objects of travel (obviously of the same type) should be unique.

For a better understanding of the complexity of the learning problem faced by ASCoL, we created metrics to learn the amount of input plans and types of plans (goal and random walk) required to effectively learn these static facts. The same set of planning instances have been used for generating both goal-oriented (GO) and random walk (RW) traces. The later have been created by using a Java-based tool able to parse a given domain model and problem and generate subsequent valid traces of a given length. Clearly, such traces usually do not reach the goal required by the planning instance, but usually provide richer information in terms of the number of transitions for different types of static facts when compared to the goal-oriented plan sequences. Goal-oriented solutions are generally expensive in that a tool or a planner is needed to generate a large enough number of correct plans to be used by the system, but they can also provide useful heuristic information.

Experimental Evaluation

Remarkable results have been achieved in complex domains, with regards to the number of static relations. We considered fifteen different domain models, taken either from IPCs[1] or from the FF domain collection (FFd)[2].

We selected domains that are encoded using different modelling strategies, and their operators include more than one argument per object type. Table 1 shows the results of the experimental analysis. A detailed interpretation of results can be found in the recent AI*IA publication (Jilani et al. 2015). All domains but Gripper, Logistics and Hanoi, exploit static relations. Input plans of these domains have been generated by using the Metric-FF planner (Hoffmann 2003) on randomly generated problems, sharing the same objects. ASCoL has been implemented in Java, and run on a Core 2 Duo/8GB processor. CPU-time usage of the ASCoL is in the range of 35-320 (ms) for each domain.

Considering a classification terminology, we can divide the relations identified by ASCoL in to four classes: true positive, true negative, false positive and false negative.

True positive These are correctly identified static relations. Relations identified by ASCoL are almost always static relations which are included in the original domain models.

True negative Dynamic relations that are (correctly) not encoded as static relations. ASCoL did not identify a static relation between arguments that are actually connected by a dynamic relation in any of the considered domains.

False positive It indicates additional relations that actually do not exist in benchmark domains. In some domains ASCoL infers one or two additional relations that are not included in the original domain model. From a Knowledge Engineering point of view, and considering the fact that such additional preconditions do not reduce the solvability of problems, such inferred relations can add value to the original model in terms of effectiveness of plan generation. This is the empirical finding limited to the domains considered for experimentation.

False negative Facts that exist and system dose not identify them. In Freecell and Gold-miner domains ASCoL does not identify all of the static relations.

[1] http://ipc.icaps-conference.org/

[2] https://fai.cs.uni-saarland.de/hoffmann/ff-domains.html

Domain	# Ops	# SR	LSR	ASR	CPU-time
TPP	4	7	7	0	171
Zenotravel	5	4	6	2	109
Miconic	4	2	2	0	143
Storage	5	5	5	0	175
Freecell	10	19	13	0	320
Hanoi	1	0	1	1	140
Logistics	6	0	1	1	98
Driverlog	6	2	2	0	35
Mprime	4	7	7	0	190
Spanner	3	1	1	0	144
Gripper	3	0	1	1	10
Ferry	3	1	2	1	130
Barman	12	3	3	0	158
Gold-miner	7	3	1	0	128
Trucks	4	3	3	0	158

Table 1: Overall results on considered domains. For each original domain, the number of operators (# Ops), and the total number of static relations (# SR) are presented. ASCoL results are shown in terms of the number of identified/learnt static relationships (LSR) and number of additional static relations provided (ASR) that were not included in the original domain model. Such relations do not compromise the solvability of problems, but prune the search space. The last column shows the CPU-time in milliseconds for finding static facts for each domain

Table 2 shows, for each considered domain, the percentages of true positive (negative) and false positive (negative) relations identified by ASCoL.

The ability of ASCoL to correctly identify static relations, that should be included as preconditions of specific operators, depends on the number of times the particular operator appears in the provided plan traces. The higher the number of instances of the operator in the plan, the higher the probability that ASCoL will correctly identify all the static relations. We now discuss some of the most interesting results.

Conclusion and Future Goals

Intelligent agents solving problems in the real-world require domain models containing widespread knowledge of the world. Synthesising operator descriptions and domain specific constraints by hand for AI planning domain models is time-intense, error-prone and challenging. To alleviate this, automatic domain model acquisition techniques have been introduced. For example, the LOCM system requires as input some plan traces only, and is effectively able to automatically

Domain	TP	TN	FP	FN
TPP	100.0	100.0	0.0	0.0
Zenotravel	100.0	66.6	33.3	0.0
Miconic	100.0	100.0	0.0	0.0
Storage	100.0	100.0	0.0	0.0
Freecell	70.0	100	0.0	30.0
Hanoi	100.0	0.0	100.0	0.0
Logistics	100.0	0.0	100.0	0.0
Driverlog	100.0	100.0	0.0	0.0
Mprime	100.0	100.0	0.0	0.0
Spanner	100.0	100.0	0.0	0.0
Gripper	100.0	0.0	100.0	0.0
Ferry	100.0	50.0	50.0	0.0
Barman	100.0	100.0	0.0	0.0
Gold-miner	33.3	100.0	0.0	66.6
Trucks	100.0	100.0	0.0	0.0

Table 2: For each considered domain, the percentage of true positive (TP), true negative (TN), false positive (FP) and false negative (FN) static relations identified by ASCoL.

encode the dynamic part of the domain model. However, the static part of the domain, i.e., the underlying structure of the domain that can not be dynamically changed, but that affects the way in which actions can be performed â is usually missed, since it can hardly be derived by observing transitions only.

In this research we briefly present ASCoL, a tool that exploits graph analysis for automatically identifying static relations, in order to enhance planning domain models. ASCoL has been evaluated on domain models generated by LOCM for the international planning competition, and has been shown to be effective.

We are considering several paths for future work. Grant, in (Grant 2010), discusses the limitations of using plan traces as the source of input information. ASCoL faces similar difficulties as the only input source to verify constraints are sequences of plans. We are also interested in extending our approach for considering static relations that involve more than two arguments In particular, we aim to extend the approach for merging graphs of different couples of arguments. Finally, we plan to identify heuristics for extracting useful information also from acyclic graphs.

References

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013a. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review* 28(02):195–213.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013b. Acquiring planning domain models using locm. *The Knowledge Engineering Review* 28(02):195–213.

Grant, T. 2010. Identifying Domain Invariants from an Object-Relationship Model. *PlanSIG2010* 57.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. 20:291–341.

Jilani, R.; Crampton, A.; Kitchin, D. E.; and Vallati, M. 2014. Automated Knowledge Engineering Tools in Planning: State-of-the-art and Future Challenges. In *The Knowledge Engineering for Planning and Scheduling workshop (KEPS)*.

Jilani, R.; Kitchen, D.; Crampton, A.; and Vallati, M. 2015. Learning static constraints for domain modeling from training plans. In *the 14th Conference of the Italian Association for Artificial Intelligence (AI* IA 2015)*, 31.

Shah, M.; Chrpa, L.; Jimoh, F.; Kitchin, D.; McCluskey, T.; Parkinson, S.; and Vallati, M. 2013. Knowledge engineering tools in planning: State-of-the-art and future challenges. In *Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling*.

Shahaf, D., and Amir, E. 2006. Learning partially observable action schemas. In *Proceedings of the national conference on artificial intelligence (AAAI)*.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. 2007. Planning domain definition using gipo. *The Knowledge Engineering Review* 22(02):117–134.

Vaquero, T. S.; Romero, V.; Tonidandel, F.; and Silva, J. R. 2007. itSIMPLE 2.0: An Integrated Tool for Designing Planning Domains. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 336–343.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence* 171(2):107–143.

Zhuo, H. H. 2015. Crowdsourced Action-Model Acquisition for Planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Using GORE method for Requirement Engineering of Planning & Scheduling

Javier Martnez Silva
Department of Mechatronics Engineering
University of São Paulo, São Paulo, Brazil,
Professor Morais, 2231

Abstract

The growing interest in the automated planning community seeks for new and better results for real applications. Requirements' analysis is a key issue over design and needs to be enhanced to fit users and stakeholder expectations. Such scenario make the researchers to focus on Knowledge Engineering (KE) applied in elicitation of planning problems and domains - mainly the early stage of the process. This proposal introduces the applying of GORE method for Engineering Requirement of planing domain modeling.

Introduction

The Knowledge Engineering for planning automated collects contributions of crucial works of researchers such as Prof. Thomas Lee McCluskey, who since the early 90s began to publish papers in this field (McCluskey and Porteous 1993). Researching in design process for building knowledge models in real fields with high quality and reliability (McCluskey 2002), and the study of requirement engineering methods applied with existing automated planning techniques are key objectives of this field (Vaquero et al. 2013).

In this scope, O-Plan was one of precursor tool in acquisition and modeling knowledge of planning on an tasks-driven approach. Most recent distribution is a web service, which is used in a wide range of dependent-domain applications (Tate and Dalton 2003).

SIPE involves an ACT (Myers and Wilkins 1997) approach, in which a system is able to give response to events in real time by performing a best possible action. Modeling of all knowledge required to generate plan is possible while external events are occurring.

Both, O-Plan and SIPE planners are predecessors for GIPO, one of the planners registered in literature with mechanisms for acquiring and modeling knowledge of independent-domain applications (Simpson et al. 2001). It address to the syntactic and semantic verification of models, improving the performance of planners; the import and export from domain definitions to PDDL format; integration of planning algorithms jointly with its execution and simulation provides a friendly environment for users. Graphical representation of dynamic objects through state machines (Simpson 2005) (Simpson 2007) is allowed since version III.

ItSIMPLE is another tool based an object-oriented approach helping to designers achieve a detailed model of the domain (Vaquero et al. 2007). The must relevant contribution is the use of Unified Modeling Language (UML) providing diagrams such as use cases, classes, state machines, time and objects. Classes, properties, relationships and constraints are defined in the class diagram, thus are modeling static characteristics of the domain.

Table 1 shows PDDL as domain modeling language by the most of the tools.

Tools	Domain model
DISCOPLAN	PDDL
EUROPA	Action Notation Modeling Language
GIPO	PDDL
FlowOpt	Work-Flow Modeling
itSIMPLE	UML
JABBAH	Business Process Management Notation
ModPlan	PDDL
VIZ	Non-Standard graphical diagrams

Figure 1: Main tools and its approach for modeling problem domain

EUROPA (Extensible Universal Remote Operations Planning Architecture) (Barreiro et al. 2012) was designed to support planning for complex systems, such as spacecraft and rovers, combining two abstractions: Constraints Satisfaction Problems and networks for modeling Simple Temporal Problems. Domain of real problems is modeled by structures called Objects with ANML (Action Notation Modeling Language) as input language on a strategy state / activity (Smith, Frank, and Cushing 2008).

FlowOpt model processes over a workflow approach simplified guiding users to create correct models (Bartak and Cepek 2008); JABBAH combines modeling using BPMN language (Business Process Management Notation) with a workflow approach (Gonzalez 2009); and finally VIZ with a graphic language simple non-standard for describing the planning areas. All these tools enable translation of domain

models to standard PDDL language (Vodrazka and Chrpa 2010).

Even PDDL being the must used language, PLEXIL (Plan Execution Interchange Language) is another language which is originally developed as a collaborative effort between NASA researchers and Carnegie Mellon University, for plans representation on real or simulated systems, in robotics, automation of operations in human habitats and systems involving intelligent software agents.(Biatek et al. 2014).

Of all these tools, itSIMPLE was the first to introduce requirement engineering techniques applied to planning problems (Vaquero and Silva 2009): requirements and relevant knowledge of the different viewpoints involved are represented by the diagram of use cases, a semi-formal representation of the UML language (OMG 2005).

GORE methods

Goal-Oriented Requirement Engineering (GORE) is a subarea of Requirement Engineering (RE), which addresses using of goals for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying requirements (Van Lamsweerde 2000).

In the literature are registers a wide number of *goal* definitions: Goals as high level objectives of business, organization or system; they capture the reasons why a system is needed and guide decisions in various levels within the enterprise (Potts, Takahashi, and Antón 1994). A goal is a condition or state which engaging issues of the world achieved by an agent (Van Lamsweerde 2000)(Yu, Dubois, and Mylopoulos 1995). According to (Lamsweerde 2004) a goal is a prescriptive statement declaring the purpose of some (existing or to-be) systems whose satisfaction generally emerged from collaboration between agents with some responsibility over the system. These goals guide requirement elaboration process resulting in the definition of domain-specific requirements.

Goals cover different kinds of concerns: functional concerns associated with services to be requested, and non-functional concerns with quality of service (safety, security, accuracy, performance, and so forth) (Chung et al. 2012). Also goal issues are defined in Artificial Intelligence field - specifically in classical planning & scheduling problem in which solution is a sequence of actions (plan) that end in a state entailing some goal previously defined (Russell and Norvig 2010).

GBRAM [Bib19], I* framework (Potts, Takahashi, and Antón 1994), NFR (Chung et al. 2000), KAOS (Van Lamsweerde 2001), Goal/Strategy Map (Bider et al. 2005), GLR (Grigorev and Kirilenko 2013) are some methods based on GORE. Of these, KAOS and I* are the most cited.

KAOS method

KAOS approach is an goal-oriented implementation of GORE method which involves a rich set of formal analysis techniques based on Linear Time Logic (LTL). Indeed, KAOS stands for *Keep All Objectives Satisfied*(Lamsweerde 2009), describing a multi-paradigm framework that combines different levels of reasoning: semi-formal, for modeling and structuring goals; and formal, based in the linear time logic formalism. Therefore, KAOS combines a semantic net of basic concepts such as assumptions, operations, objects and agents, with linear time logic(Lamsweerde 2009).

Basically, KAOS is a goal-driven elaboration method that provides a specification language for capturing *WHY*, *WHO* and *WHEN* aspects in addition to the usual *WHAT* requirements.

Graphically, goals are represented in KAOS diagram by parallelograms, while requirement borders are drawn in bold line and agents are represented by hexagons as in Fig.2, goal diagram for block world problem (Russell and Norvig 2010).

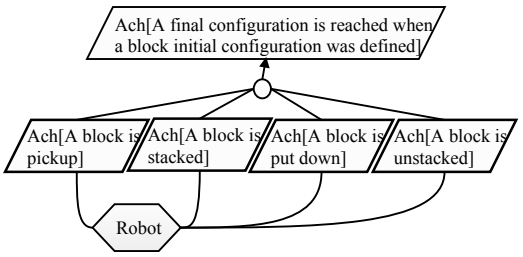


Figure 2: Goal Diagram of simple block world

I* method

I* method is a conceptual modeling technique for modeling and reasoning organizational environments and their information systems introduced by (Yu 1994). Strategic Dependency and Strategic Rationale models are key models of I* approach. Strategic Dependency model describes relationships of dependency among various agents over organizational environment. Strategic Rationale model describes stakeholders' concerns and viewpoints over the system and environment.

Fig.3 shows Strategic Dependency diagram of the same block problem. Analyzing of this model, shows how actors are key strategic into I* for representing motivations, intents and rationale behind actions to achieve goals. Next, we comparing both approaches with focus on modeling non functional requirements (NFR); understanding a boundary between system and it environment; modeling of objects involved in knowledge domain and actor's concerns.

KAOS versus I*

NFR in KAOS are mainly treated as goals and I* models as soft-goals which allowing for qualitative reasoning. Same mission of goal model in KAOS, help to clarify a boundary between a software-to-be and its environment. This model is organized in a tree where leaves are assignable to single agents (software or human). If leaves are assigned to the software-to-be, model as a requirement and if is assigned to environment agent is an assumption, respectively.

Objects related with knowledge domain can be modeled through of one of KAOS diagrams: Object Diagram. In I*

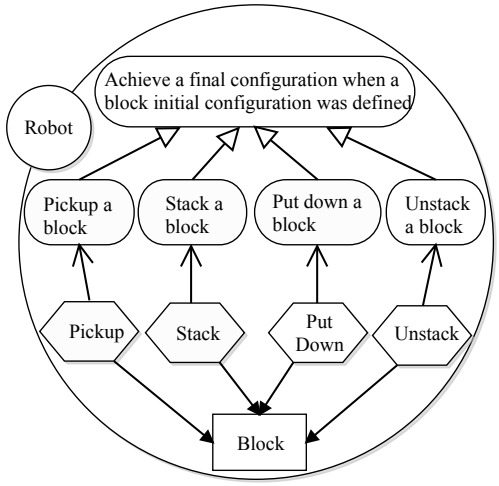


Figure 3: Strategic Dependency Diagram of simple block World

are represented by resources as dependence between agents.

Fig.4 shows Object Diagram from goals modeled in Goal Diagram.

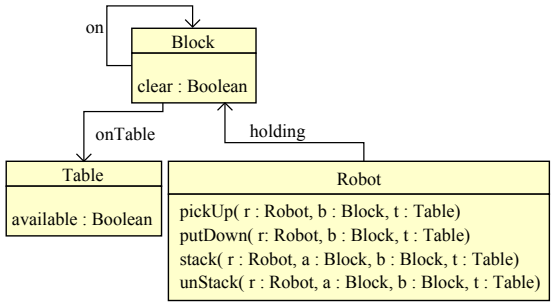


Figure 4: Object Diagram of simple block world

Another advantage of KAOS method is its potentiality of express to behavioral goals a formal representation in Linear Time Logic (LTL) formalism:

Goal Achieve A block is stacked
Def. Formal $\forall(a, b : Block)$
$[stacked(a, b) \rightarrow \Diamond(On(a, b),$
$b.clear = false)]$

Where goals are expressed formally as:

$$C \Rightarrow \Theta T$$

C is the current condition, T is the target condition and Θ is one of the LTL operators represented in Table1.

These operators can be quantified by a time stamp d , so that $\Diamond_d$ means *eventually* in the future before deadline d, and $\Box_d$ means *always* in the future up to deadline d.

Table 1: Temporal Logic Operators

Operator	Description
◯	In the next state
♢	Eventually in the future
□	Always in the future

GORE for planning & scheduling

A key challenge aims to modeling the planning & scheduling problems features through requirements using formal methods - even for medium and large problems - with a schematic language, a first stage of eventual verification and validation from initial models to a consistent model transferable to automatic planners in a later stage.

Our proposal is to provide a clear process for design early stage in which modeling of requirements is a systematic process using GORE method, considering time constraints (such as the duration of sub-processes) and methods for requirement analysis using Petri nets, guaranteeing a consistent input to the automated planners.

Fig.5 provides an overview of steps to follow on the design of planning problem proposed by (Vaquero et al. 2013).

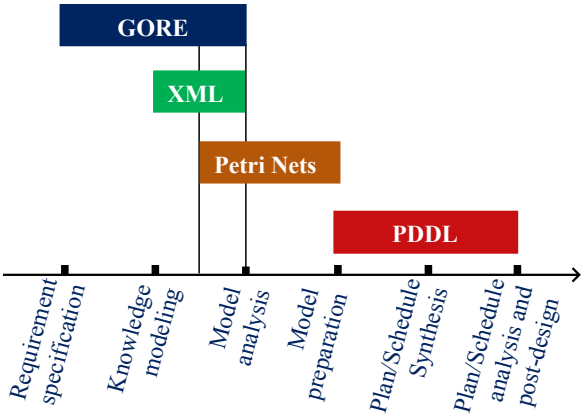


Figure 5: How is added GORE methods over early design phases of planning problems proposed by (Vaquero et al. 2013)

Over these phases, we propose GORE for requirement specification and knowledge modeling stages, and Petri Net formalism in the later stage to modeling and analyzing operations associated to certain goals. XML is proposed for parsing and as starting point in obtaining of PNML(XML alternative for Petri nets).

Improving the design of problems in planning & scheduling- specifically on the early steps of elicitation and analysis of requirements- using GORE (Goal Oriented Requirement Engineering) will provide a comparison study - in terms of formalization - with other approaches as UML.

References

Barreiro, J.; Boyce, M.; Do, M.; Frank, J.; Iatauro, M.; Kichkaylo, T.; Morris, P.; Ong, J.; Remolina, E.; Smith, T.; et al. 2012. Europa: A platform for ai planning, scheduling, constraint programming, and optimization. In *Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12)–The 4th International Competition on Knowledge Engineering for Planning and Scheduling*.

Bartak, R., and Cepek, O. 2008. Nested precedence networks with alternatives: Recognition, tractability, and models. *Artificial Intelligence: Methodology, Systems, and Applications* 235–246.

Biatek, J.; Whalen, M. W.; Heimdahl, M. P.; Rayadurgam, S.; and Lowry, M. R. 2014. Analysis and testing of plexil plans. In *Proceedings of the 2nd FME Workshop on Formal Methods in Software Engineering*, 52–58. ACM.

Bider, I.; Johannesson, P.; Nurcan, S.; Etien, A.; Kaabi, R.; Zoukar, I.; and Rolland, C. 2005. A strategy driven business process modelling approach. *Business Process Management Journal* 11(6):628–649.

Chung, L.; Nixon, B.; Yu, E.; and Mylopoulos, J. 2000. Nfr in software engineering.

Chung, L.; Nixon, B. A.; Yu, E.; and Mylopoulos, J. 2012. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media.

Gonzalez, A. 2009. JABBAH: A Java application framework for the translation between business process models and HTN. *Association for the Advancement of Artificial Intelligence* 28–37.

Grigorev, S., and Kirilenko, I. 2013. Glr-based abstract parsing. In *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, 5. ACM.

Lamsweerde, A. V. 2004. Elaborating security requirements by construction of intentional anti-models. *Proceedings. 26th International Conference on Software Engineering* (May):148–157.

Lamsweerde, A. v. 2009. *Requirements Engineering: From System Goals to UML Models to Software Specifications*, volume I. Wiley.

McCluskey, L., and Porteous, J. M. 1993. Two Complementary Techniques in Knowledge Compilation for Planning. *In Proceedings of the 3rd International Workshop on Knowledge Compilation and Speedup Learning*.

McCluskey, L. 2002. Knowledge Engineering: Issues for the AI Planning Community. *Proceedings of the AIPS-2002 Workshop on Knowledge Engineering Tools and Techniques for AI Planning, Toulouse, France*.

Myers, K. L., and Wilkins, D. E. 1997. The Act Formalism. *SRI International Artificial Intelligence Center*.

OMG. 2005. UML 2.0 OCL Specification.

Potts, C.; Takahashi, K.; and Antón, A. I. 1994. Inquiry-based requirements analysis. *IEEE software* (2):21–32.

Russell, S., and Norvig, P. 2010. *Artificial Intelligence. A Modern Approach*. Pearson Education, Inc, third edit edition.

Simpson, R.; McCluskey, L.; Zhao, W.; S, A. R.; and Doniat, C. 2001. GIPO: an integrated graphical tool to support knowledge engineering in AI planning. *Proceedings in European Conference on Planning (ECP-2001)*.

Simpson, R. M. 2005. Gipo graphical interface for planning with objects. *International Competition on Knowledge Engineering for Planning and Scheduling* 34–41.

Simpson, R. 2007. Structural domain definition using GIPO IV. *In Proc. 2nd Int. Competition on Knowledge Engineering for Planning and Scheduling* 3(Hoffmann).

Smith, D.; Frank, J.; and Cushing, W. 2008. The anml language. *Proceedings of ICAPS-08*.

Tate, A., and Dalton, J. 2003. O-Plan: a Common Lisp planning web service. *Proceedings of the International Lisp Conference*.

Van Lamsweerde, A. 2000. Requirements engineering in the year 00: a research perspective. In *Proceedings of the 22nd international conference on Software engineering*, 5–19. ACM.

Van Lamsweerde, A. 2001. Goal-oriented requirements engineering: A guided tour. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, 249–262. IEEE.

Vaquero, T. S., and Silva, R. 2009. From Requirements and Analysis to PDDL in itSIMPLE3.0. *Proceedings of the ....*

Vaquero, T.; Romero, V.; Tonidandel, F.; and Silva, R. 2007. itSIMPLE 2.0: An Integrated Tool for Designing Planning Domains.

Vaquero, T. S.; Beck, J. C.; McCluskey, L.; and Silva, R. 2013. Knowledge Engineering for Planning & Scheduling: Tools and Methods. 1:1–15.

Vodrazka, J., and Chrpa, L. 2010. Visual design of planning domains. *In: Proceedings of ICAPS 2010 workshop on Scheduling and Knowledge Engineering for Planning and Scheduling (KEPS)* 2–3.

Yu, E.; Dubois, E.; and Mylopoulos, J. 1995. From organization models to system requirements. a cooperating agents approach. In *in: 3rd Intl. Conf. on Cooperative Inf. Sys.(CoopIS-95*. Citeseer.

Yu, E. 1994. *Modeling strategic actor relationships for process reengineering*. Ph.D. Dissertation, PhD Thesis, University of Toronto.

Critical Constrained Planning and an Application to Network Penetration Testing

Marcel Steinmetz
Saarland University
Saarbrücken, Germany
{steinmetz}@cs.uni-saarland.de

Abstract

Critical constrained planning is a very interesting, though also very under explored class of classical, as well as probabilistic, planning. In a broad view, constraints allow to limit the space of solutions to solutions that satisfy certain kinds of conditions. In this work, we are going to develop techniques that are in particular suited to solve critical constrained planning instances. Automated, and semi automated network penetration testing has gotten a rise in attention in the previous decade due to the growing size of todays networks: it is not possible anymore to manually check reasonably sized networks for security threats. Planning overall, and in particular critical constrained planning appears to be a very fruitful direction in this area, although many of the previous works lack of either an accurate representation of the problem, or they lack of scalability. In this work, we will continue on Hoffmann's (2015) taxonomy model, and we will identify tractable fragments of, and we will develop efficient methods to solving the variant classes of network penetration testing.

Introduction

Critical constrained planning problems impose additional hard requirements on the solutions of the problem. We will in particular focus on resource constrained planning problems (Nakhost et al. 2012), so problems that contain some sorts of consumable resources (time, money, ...) that cannot, or only sparsely, be refilled. Although there are many works that consider problems facing resource consumption (e. g., (Koehler 1998), (Haslum and Geffner 2001)), only a very few consider the case of a limited resources that keeps decreasing as the system progresses (e. g., (Nakhost et al. 2012), (Hoffmann et al. 2007), (Gerevini et al. 2008)). Resource constrained planning problems model a bunch of interesting real world applications. For instance, when modeling time as resource, then typically this resource cannot be reset. Other examples are a fixed money budget, or space agents with disconnected (or inactive) power supply. Even though resource constrained planning problems are only a subclass of general resource planning problems, they offer many challenges that make resource constrained planning on its own very interesting. In general, due to the non-increasing structure of the resource, the planner has to plan far ahead in order to not run out of resources before reaching the goal. Nakhost et al. have shown that with growing *resource constrainedness*, current state of the art methods are getting more and more trouble solving these problems (Nakhost et al. 2012). Our work will be focusing around the following observation: a typical phenomenon in resource constrained planning problems is the high density of dead end states – as the search progresses, more and more solutions are ruled out. Thus, dead end detection will be an important key to solve these kinds of problems efficiently.

On the other hand, automated network penetration testing (Arce and McGraw 2004), an apriori completely unrelated area to critical constrained planning, will constitute the second major part of our work. Due to the growing size of today's networks, it is getting harder (and in medium to large company networks even impossible) to identify security threats by hand. Rather, companies are using automated, and semi automated tools to analyse vulnerabilities of their networks. Using planning to simulate attacks to networks is a very promising future direction in (semi-) automated network security testing. A company called CoreSecurity (*http://www.coresecurity.com/*) is already commercially using a planner inside one of their tools to generate possible *attack plans* that are provided to an human security officer to guide the attention to particular, possible security threatening, regions of the network (Lucangeli et al. 2010). One problem of this approach is that it requires a global and exact model of the network, including all the network host configurations. In practice, this is clearly impossible to get and to maintain. Ideally, the model should start with minimal possible knowledge about the network and host configurations. This idea was followed by Sarraute et al. in their design of network penetration testing as solving POMDPs (Sarraute et al. 2011; Sarraute et al. 2012). Unfortunately, solving such POMDP models is only feasible for a very small number of hosts (Sarraute et al. 2012), and thus does not scale to real world networks. In this work we will continue with Hoffmann's work on finding feasible, yet realistic models of penetration testing (Hoffmann 2015). One very typical aspect of probabilistic models of penetration testing is that the probability of reaching a goal state will be low. In other words, the probability of reaching dead ends will be very high. So, dead end detection will play an important role to solve penetration testing problems efficiently, as well.

Learning to Recognize Dead Ends

Current state of the art heuristic search based planning approaches do not really consider the problem of dead end detection specifically. Rather, they make use of the *artifact* that when using a *safe* heuristic function h, whenever $h(s) = \infty$, then s is a dead end. If a state s is identified as dead end, it is not further considered in search (we say that s is *pruned*). Recently, a few works have been published on the topic of detecting planning problem *unsolvability* (e. g., (Bäckström et al. 2013; Hoffmann et al. 2014)). Although these techniques are designed to prove the unsolvability of a planning task, they still can be used in solvable problems: we can use them in order to identify dead ends during the search. This can lead to a significant performance advantage in problems where the number of dead ends abound (Steinmetz and Hoffmann 2016). However, these *unsolvability heuristics* are typically computed in a preprocessing step. In our work we will follow a completely different direction. Instead of computing a data structure detecting dead ends before the search, we make use of the information becoming available during the search in order to constantly refine our dead end identifier, and thus detecting more dead ends.

Observe that the search itself gives proofs of dead ends while exploring the state space (Steinmetz and Hoffmann 2016). Whenever a state s is encountered in an open, closed list search so that each state of the search space that can be reached through s, and s itself are closed, then we know that s must be a dead end. Given this information and additionally $u(s) < \infty$, for an unsolvability heuristic u, we can refine u so that s becomes recognized under u (provided that there is a refinement algorithm for u that supports this). After this refinement, u might *generalize* to other, not yet seen, dead ends as well. For $u = h^C$ (Keyder et al. 2014; Hoffmann and Fickert 2015), this generalization happened in a large scale in the three available resource constrained benchmark domains (NoMystery, Rovers, and TPP) (Steinmetz and Hoffmann 2016). In those experiments, the search space reduction is tremendous. The geometric mean is around two orders of magnitude, and the reduction goes even up to 5 orders of magnitude. However, as the size of the set of atomic conjunctions C is continuously growing, the computation of h^C is getting computationally more complex the more dead ends are being learned. To really benefit from the search space reduction, one has to reduce the number of calls to h^C. To prefilter the calls to h^C, we extract a clause ϕ so that for all t with $t \not\models \phi$ it is $h^C(t) = \infty$, after each time when we have evaluated h^C on a state s and $h^C(s) = \infty$. Inspired by SixthSense (Kolobov et al. 2012), we compute ϕ based on greedily minimizing $\phi = \neg s$ while keeping $h^C(s) = \infty$. This already works pretty well, up to 98% (in the geometric mean) of the dead ends recognized by h^C could be filtered through these conjunctions.

In future work, we have to address open questions in three different parts of the approach: (1) search algorithm, (2) h^C dead end detection, and (3) finding other unsolvability heuristics u that can be used in the framework. (1) Our current experiments have shown that only very few dead ends are learned when running optimal search (A* (Hart et al. 1968)) instead of a depth first exploration of the state space. The intuitive reason is that the farer the distance to the initial state, the higher is the chance of finding dead ends, and thus the more can we learn. In contrast, in A*, the exploration is biased towards the initial state, meaning that fewer dead ends become known, and thus we can learn less. This brings up the question, how can we learn more dead ends while preserving the optimality of the search? One possibly promising direction might be to use IDA*(Korf 1985) instead. What about an anytime search algorithm, such as heuristic depth first search (Bonet and Geffner 2006)?

In (2), we distinguish between (a) the h^C heuristic itself, and (b) learning clauses based on h^C. Because h^C is getting so expensive to compute in the long term (learning many dead ends), there might be room for improvements in the conjunction selection algorithm. Choosing different sets of conjunctions during the refinement has a direct effect on the detection of previously unrecognized dead ends. In our current implementation, we try to greedily minimize the size of each single conjunction that is selected during refinement. Can we choose the set of conjunctions differently so that the number of selected conjunctions becomes smaller than before while recognizing at least as many dead ends? Or does it even make sense to construct larger sets of conjunctions C, e. g.because the $|C|$ by number of detected dead ends ratio becomes smaller? A way towards answering these questions is to identify the *value* of a conjunction, i. e., in how many detected dead ends s does the conjunction play a role in obtaining $h^C(s) = \infty$. But how do we find the value of a conjunction efficiently? Another idea, borrowed from the SAT community (Goldberg and Novikov 2002), do we actually need all conjunctions ever added to the set of atomic conjunctions C? Or can we *forget* some of the conjunctions after a while? Another issue of our current refinement algorithm is that it only allows to refine the set of atomic conjunctions on states s if all of their successor states are already recognized. This in particular makes it hard to use h^C learning additional to other dead end identifiers.

Regarding (2b), recall that the clauses are checked before h^C is evaluated, and h^C is only evaluated if non of the clauses matched. As previously mentioned, actually almost all dead ends are identified through the clauses, $i. e.$, h^C is barely evaluated on dead ends. So, why evaluating h^C at all? It turns out that when just using clauses to identify dead ends, one performs significantly worse than when additionally evaluating h^C. The reason for this is simple: by always skipping the evaluation of h^C, we do not learn enough clauses to cover all the recognized dead ends. Can we select when to evaluate h^C, or when to stop evaluating h^C? Or is it even possible to learn multiple clauses at once that cover all recognized dead ends, and thus getting rid of the evaluation of h^C completely?

(3) Our current results are all based on h^C. However, there might be also other heuristic functions that allow a refinement during search and which might perform equally well, or even better.

Last but not least, can we generalize this approach to a less strict definition of *dead ends*: can we learn to identify states whose only (possible) solution is via one of its ancestors?

Network Penetration Testing

We will be investigating the different classes of Hoffmann's taxonomy models (Hoffmann 2015), and we will design efficient methods to solve them. In short, the planning model simulates an attacker whose goal is to get access to sensitive parts of the network. We will mainly focus on probabilistic models as these allow to easily encode partial knowledge about the network, and even allow to express exploits that are of stochastic nature (e. g., buffer overflow attacks). The optimization criterion that we are looking at is maximizing the probability of reaching the goal as the typical question in network security is how likely it is that an attacker gets access to sensitive areas of the network. Since finding optimal solutions in MDPs (POMDPs) is notoriously hard, we will also consider weaker objectives: finding attack policies that succeed with at least θ probability, or finding policies whose success probability differs from the optimal policy by at most δ (Steinmetz et al. 2016).

First experiments show that solving even these highly restrictive classes of probabilistic planning models is only feasible for small networks (Steinmetz et al. 2016). Due to the large number of dead ends, blindly instantiated heuristic search algorithms perform similar to state space exhaustive methods (VI) – as opposed to other domains where blindly initialized heuristic search algorithms generally perform much stronger than VI.

Unfortunately, we cannot make use of admissible heuristic functions simply because none exist. Thus, finding admissible heuristic functions for goal probability is one of our main research questions. The common approach to find admissible heuristic functions is to identify tractable fragments, i. e., fragments that allow to compute the exact goal probability in polynomial time.

Besides finding heuristic functions, we can improve the efficiency of heuristic search algorithms (as well as VI) through several state space reduction techniques. Such techniques have already been used successfully in classical planning (Pochter et al. 2011; Hall et al. 2013; Wehrle and Helmert 2012). In particular partial order reduction seems to be a very promising direction in network penetration testing: usually the search can select the next host to attack out of a large set of network hosts. However, the order in which they are attacked does not matter with respect to the probability of reaching a goal state – though, the search will still enumerate all possible permutations (an artifact of the apply-once constraint).

Finally, budget constrained network penetration testing is an interesting problem, both, from a practical as well as theoretical view point. It is relevant to network penetration testing in practice because one does not always want to consider *all* attack policies, but rather only those that can be executed in for example a reasonable amount of time, respectively by using a reasonable amount of money. This problem leads us again to the first question: how to learn to recognize dead ends during search? However, now, we are no longer considering search in a deterministic state space, but rather search in a probabilistic state space. So, can we generalize the methods developed for critical constrained (classical) planning to probabilistic problems as well?

References

Arce, I.; and McGraw, G. 2004. Why attacking systems is a good idea. IEEE Computer Society - Security & Privacy Magazine 2(4)

Bäckström, C.; Jonsson, P.; and Ståhlberg, S. 2013. Fast detection of unsolvable planning instances using local consistency. In Proc. SoCS13.

Bonet, B.; Geffner, H. 2006. Learning Depth-First Search: A Unified Approach to Heuristic Search in Deterministic and Non-Deterministic Settings, and Its Application to MDPs. In Proc. ICAPS'06.

Gerevini, Alfonso E.; Alessandro Saetti; and Ivan Serina. 2008. An approach to efficient planning with numerical fluents and multi-criteria plan quality. Artificial Intelligence 172.8 (2008): 899-944.

Goldberg, E.; and Novikov, Y. 2002. BerkMin: a fast and robust SAT- solver. In Design, Automation and Testing in Europe Conference, 142149, March 2002.

Hall, D. L. W.; Cohen, A.; Burkett, D.; Klein, D. 2013. Faster Optimal Planning with Partial-Order Pruning. In Proc. ICAPS'13.

Hart, P. E.; Nilsson N. J.; Raphael B. 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics SSC-4(2), 100-107.

Haslum, P.; and Geffner, H. 2001. Heuristic planning with time and resources. In Proc. ECP01, 121132.

Hoffmann, J.; Kautz, H.; Gomes, C.; and Selman, B. 2007. SAT encodings of state-space reachability problems in numeric domains. In Proc. IJCAI07, 19181923.

Hoffmann, J.; Kissmann, P.; and Torralba, A. 2014. "Distance"? Who Cares? Tailoring Merge-and-Shrink Heuristics to Detect Unsolvability, Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14), Prague, Czech Republic, August 2014.

Hoffmann, J.; and Fickert, M. 2015. Explicit Conjunctions w/o Compilation: Computing $h^{\mathrm{FF}}(\Pi^C)$ in Polynomial Time. In Proc. ICAPS'15.

Hoffmann, J. 2015. Simulated Penetration Testing: From "Dijkstra" to "Turing Test++". Invited paper in Proceeding ICAPS'15.

Nakhost, H.; Hoffmann, J.; Müller, M. 2012. Resource-Constrained Planning: A Monte Carlo Random Walk Approach. In Proc. ICAPS'12.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2014. Improving delete relaxation heuristics through explicitly represented conjunctions. Journal of Artificial Intelligence Research 50:487533.

Koehler, J. 1998. Planning under resource constraints. In Proc. ECAI98, 489493.

Kolobov, A.; Mausam; and Weld, D. S. 2012. Discovering hidden structure in factored MDPs. Artificial Intelligence 189:1947.

Korf, R. E. 1985. Depth-first Iterative-Deepening: An Optimal Admissible Tree Search. Artificial Intelligence 27, 97–109.

Lucangeli, J.; Sarraute, C.; and Richarte, G. 2010. Attack planning in the real world. In Workshop on Intelligent Security (SecArt 2010).

Pochter, N.; Zohar, A.; Rosenschein, J. S. 2011. Exploiting Problem Symmetries in State-Based Planners. In Proc. AAAI'11.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2011. Penetration testing == POMDP solving? In SecArt11.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2012. POMDPs make better hackers: Accounting for uncertainty in penetration testing. In AAAI12.

Steinmetz, M.; and Hoffmann, J. 2016. Towards Clause Learning State Space Search: Learning to Recognize Dead Ends. In Proc. AAAI'16.

Steinmetz, M.; and Hoffmann, J.; and Buffet, O. 2016. Revisiting Goal Probability Analysis in Probabilistic Planning. In Proc. ICAPS'16.

Wehrle, M.; Helmert, M. 2012. About Partial Order Reduction in Planning and Computer Aided Verification. In Proc. ICAPS'12.

Human-Robot Communication in Automated Planning

Aleck MacNally
amacnally@student.unimelb.edu.au
University of Melbourne
Parkville VIC 3010

Abstract

With the continued integration of artificial intelligence and robotics into human society the need for human-aware and human-in-the-loop planning becomes ever more prominent. So far research has been conducted which addresses acting in a human world to achieve joint or independent goals or to assist humans in completing their own goals. However, little research has been published that pertains to human-robot communication whilst planning. A robot's ability to communicate efficiently and effectively with humans will allow the robot to be more useful to the human, in that the human may extract necessary information, or understand the actions that the robot may perform, as well as making the robot more efficient at achieving its own goals. With these benefits in mind this doctoral research will address when, why and how information is conveyed to humans by robots during planning s well as integrating into the planning process an ongoing negotiation between a human and the planner (a robot)

Introduction

Human-robot interaction presents unique challenges in the area of Automated Planning. Research into such problems as producing safe and predictable plans, inferring a human's goals from their actions and robot-human communication is on-going.

It is important when dealing with any form of human-robot interaction and human-aware planning that we understand the scenario between the robot and the human (Shah et al. 2011; Talamadupula et al. 2014; Chakraborti et al. 2015a; Levine and Williams 2014). These scenarios can be categorized as follows:

- The human is omnipotent
- The robot is omnipotent
- The robot and human plan separately but for a joint goal
- The robot and human plan separately for independent goals but must interact due simply to being in the same environment.
- The robot may adjust their goals to assist humans

The manner in which the challenge of communicating between humans and robots is addressed, will differ depending on which scenario the system is in. For instance when the human is omnipotent no information needs to go from the robot to the human. Whereas when humans and robots must work together to achieve a joint goal the communication may include negotiations with the human to produce joint plans which are less costly than working independently.

This doctoral research will address communication in two different areas of human-robot interaction. The first area is the communication of plans and goals to a human. This is important both when humans and robots work independently as well as when they share a joint goal. The second area of research is in negotiations between humans and robots, which is important when robots and humans work toward a joint goal.

Related Work

Shah et al. implemented the CHASKi executive which focused on a joint goal in a human-robot team. The research took inspiration from human-human interaction and used communication in a limited way to indicate when each agent was committing to a subtask or had completed one. CHASKI also allowed humans to issue requests to the robot.

Chakraborti et al. implemented a system which used the idea of resource profiles which temporally tracked the usage of specific resources. They constrained their resulting plan to minimize the overlap between these profiles. In this framework agents were able to negotiate for resources when there was no feasible plan. Similarly, negotiations were addressed by Karpas et al. who introduced temporal uncertainty to the PIKE executive (Levine and Williams 2014) by negotiating with the user to relax temporal constraints.

Unhelkar and Shah put forward CONTACT, a system handling multi-agent settings that decides whether an agent should communicate information based upon a reward system and an approximate idea of what is known by all other agents in the system.

Implicit human-robot communication has been studied by Zhang, Zhuo, and Kambhampati who used Machine Learning techniques to label the degree of "social acceptability" of plans

Human-Robot Communication

In this extended abstract two aspects of human-robot communication will be addressed. First, the communication of a

robots plans to the humans involved in the system. Without loss of generality, goals shared between humans and robots can be deemed to be independent. Second, the use of communication to negotiate between humans and robots, which will directly relate to humans and robots completing joint goals.

Communicating Plans

As robots become more prevalent in human society it will become necessary to ensure that they act in a manner which is understandable and predictable to the humans with whom they are cohabiting (Alterovitz, Koenig, and Likhachev 2014). To this end it is important that a plan which may seem strange to a human is explained properly by the robot agent. Stubbs, Wettergreen, and Hinds underlined this need when they found that humans working with robots were less productive when the autonomy of the robots was increased and their 'common ground' (shared knowledge) decreased. This introduces three challenges: which plans require explanation, when is an agent allowed to communicate their plan and what does an agent communicate when communicating a plan?

The first question follows from the assumption that a robot communicating all their plans to a human would become a nuisance. Therefore it is necessary that only plans which require explanation be explained. Once the robot has selected a plan to explain, this explanation must be heard and must not overlap any communication currently occurring and must also not happen when the human is completing a task which requires concentration such as driving. This is covered by the second question. The last question addresses how much information is necessary to convey to a human agents once a plan is to be explained. It is assumed that the optimal amount of information required to fully explain the plan to the human is the minimal amount, relating to the assumption from the first question.

Determining whether to communicate a goal It is important that a robot's goals are not ambiguous. This means that a human should be able to infer a robot's goals from its actions. To that end it is considered preferable to have a robot which acts obviously toward a single goal.

In many cases the nature of the environment and model of the robot result in plans which are optimal yet ambiguous. This ambiguity is formalised in the concept of the worst case distinctiveness (wcd). The wcd represents the size of the largest shared prefix of two or more optimal plans leading to different goals (Keren, Gal, and Karpas 2014). The larger the wcd, the larger the ambiguity inherent in the environment and robot's model. When the wcd of the model is too large it is necessary for the robot to communicate their goal to the human.

The idea of our approach is that a human is watching a robot, but not attempting to achieve any goals of its own. The human uses goal recognition as set out by Ramirez and Geffner in order to infer the goal of the robot. The robot uses the wcd of the system to determine if the human will infer its goals correctly or if it will need to communicate its goals. We formalize this idea below.

The robot is given a planning problem $\Pi =< F, I, A, G >$, and has a set $\mathcal{G}$ of the possible goals, where $G \in \mathcal{G}$. The human in this case has knowledge of $\mathcal{G}$ as well as Π except for G. The human will not plan itself but will reason using goal recognition. Using the framework published by Keren, Gal, and Karpas it is possible to find the worst-case distinctiveness (wcd) of the problem. It is put forward that if the $wcd > \alpha$, then for any plan π the robot must communicate G to the human, otherwise G is deemed obvious enough to the human from π. The quantity α is yet to be established.

A more computationally intensive solution to this problem is that as we know the current goal of the robot, G, we can calculate:

$$wcd_G = \max_{G' \in \mathcal{G}, G' \neq G} wcd(\Pi, \{G, G'\})$$

where $wcd(\Pi, \{G, G'\})$ refers to the wcd of the problem with the possible goal set consisting of just G and G'. The wcd_G represents the worst case distinctiveness between each goal and G. The wcd_G has wcd as an upper-bound, so this method will never force the robot to communicate more than in the above method. Using wcd_G instead of wcd will reduce the amount of times that a robot will have to communicate with the human, however, it will require the robot to perform $O(|\mathcal{G}|)$ calculations of a wcd.

To improve upon this idea it would be advantageous to model the human belief in the goal of the robot as well as the human's own goals. Doing this would allow one to reason more accurately as to when the human might need the knowledge about what goal the robot is pursuing in order to pursue their own goals.

Determining whether to communicate a plan Given a situation in which the human has knowledge of the goals of the robot, it is preferable that a plan which is to be executed is predictable and familiar to the human. If it is not predictable or familiar, the plan must be communicated. Ensuring this will make robots more trustworthy to humans. The degree to which a given plan is surprising will be determined by combining in some suitable way the wcd measures for every sub-goal achieved by the plan. This will be addressed in future research.

Determining when an agent should communicate Once a robot has found it necessary to communicate a plan due to either the ambiguity in the robots goals or in its plan, the problem becomes determining when to communicate the plan. In particular in problems which relate to communicating over a channel which is being used by multiple agents and is affected by external sources. Assuming an axiomatization of communication actions over a channel as well as the ability to observe the state of the channel, this problem can be modeled as an MDP in which the state of the channel is modeled as observations and an agent can choose to communicate with respect to these observations. this idea will be explored in further research.

The second problem in this section is making sure that a communication action does not interrupt a critical human task. Such task are those which would jeopardize the

safety of humans if they were interrupted such as driving or surgery. A simple solution to this problem is to have a human action which locks communication and a human action which unlocks communication. A second solution is to use a temporal multi-agent planner with mutual exclusion between actions which require concentration and robot communication actions. This topic will also be addressed in future research.

Determining the minimum amount of information to communicate The last question which will be addressed in this section is "how much information in a robots plan needs to be communicated?". A robot's goals will consist of a conjunction of logical statements. When these goals are created by a human they can be named easily, and then this name can be produced when it is determined that the goal is to be communicated. In the case where a robots goals are automatically generated such as in (Chakraborti et al. 2015a), the information within these goals must be summarized in a way which is human understandable. This is equally true for a plan which needs to be communicated.

A possible scheme for determining which actions of a plan need to be communicated to a human is described next. Given that a robot has an optimal plan π for a goal G, where the human has knowledge of G, we suggest that the optimal amount of information that needs to be communicated is the minimal subsequence $\pi_m \subset \pi$ (that is minimal in length not cost) which will render $P(G|\pi_m) = 1$. It is assumed that if from this π_m the human can calculate that the goal is G than, even if the plan is not human-intuitive, it is understood by the human. This will be addressed in future research.

Negotiating

Negotiations between humans and robots has already been attempted in various ways such as in Talamadupula et al. where negotiations were made over resources and in Karpas et al. where time was negotiated with. In this section we utilize the idea of negotiation to allow a problem to be solved by a joint human-robot team where the model of the robot is known but the model of the human is not. Using this idea the robot constructs a partial model of the human throughout the negotiations.

The idea of this approach is that an agent must solve a problem optimally and use the assistance of a human with a fixed communication penalty rate. To this end the agent assumes that the human may achieve any fact and after each iteration of communication, the robot updates the model of the human and then re-plans until there is a satisfying plan that both the human and robot agree is possible.

For a classical planning problem $\Pi =< F, I, A, G >$, where A is the robot's action set, we compile the model of the human into the robots model by augmenting it as follows. Let us consider the initial planning problem for the first iteration: $\Pi' =< F, I, A_0, G >$, where:

$$A_0 = A \cup \{a_h^f, a_h^{\neg f} | f \in F\}$$

where a_h^f is an action with:

- $prec(a_h^f) = \{\neg f\}$
- $add(a_h^f) = \{f\}$
- $del(a_h^f) = \varnothing$
- $cost = \lambda \times \sum_{a \in A} cost(a)$

Where $a_h^{\neg f}$ is defined similarly. a_h^f encodes the assumption that the human h can make f true. With these additional actions the robot can achieve all facts in the problem. The λ is a constant which encodes a fixed penalty for asking for help. A high λ would produce and scenario where the robot is expected to work on its own and to only seek help when it could not find a satisfying plan. If it is intended that the robot and human coordinate more than λ should be set lower.

Following the initial stage of planning the robot presents the human with the actions that she must complete (if there are any). She may then do one of two things: accept the plan as it is or negotiate with the robot. If she accepts then the problem is solved. If she chooses to negotiate, she presents the robot with additional knowledge about her model. For instance she may say that she cannot complete a_h^f but can complete $a_h^f\prime$ where $a_h^f\prime$ is equal to a_h^f with a different cost. She could also provide the robot with her entire set of actions if she chose to. With this framework the human may give as little or as much information to the robot as the human wishes. When the human has decided what information she is going to impart to the robot, she places the actions she cannot complete in $A_j^{impossible}$, this would include a_h^f from the example above, and all the actions she can complete in A_j^{poss}, which would include $a_h^f\prime$, where j is the current iteration of negotiation. The robot's action set is updated as follows:

$$A_j = \{A_{j-1} \setminus A_j^{imposs}\} \cup A_j^{poss}$$

The robot will then re-plan with the new set of actions A_j and negotiate until the human accepts the plan, or the robot cannot find a plan.

It is important to note that with this approach the model of what the human can do is not required. The robot assumes that the human can do anything and each turn the robot updates its understanding of the human's model with partial information given to it by the human

This approach will produce an optimal plan for the robot and for what the robot believes the human can do, however without a complete model for what the human can do an optimal plan for both agents cannot be found.

Conclusion

This extended abstract addressed two areas of communication between humans and robots. The first was the problem of communicating plans and goals to humans either in the same environment or to those completing the same goals. This involved three distinct challenges which were which plans and goals should a robot communicate, when should a robot communicate them and how much information was required to communicate a goal or plan? A solution to the first

question was put forward which involved the computation of the worst case distinctiveness of the system (Keren, Gal, and Karpas 2014). The second area of research in this paper addresses negotiation. A scheme for negotiations between a robot and human was put forward which involved compiling a partial human action model and a planning problem into a classical planning problem which was then solved. The solution was presented to the human and then the partial human model was update. This process continues until a solution is found.

References

Alterovitz, R.; Koenig, S.; and Likhachev, M. 2014. Robot planning in the real world: research challenges and opportunities. *National Science Foundation*.

Chakraborti, T.; Briggs, G.; Talamadupula, K.; Zhang, Y.; Scheutz, M.; Smith, D.; and Kambhampati, S. 2015a. Planning for serendipity. *Intelligent Robots and Systems (IROS)* 5300–5306.

Chakraborti, T.; Zhang, Y.; Smith, D.; and Kambhampati, S. 2015b. Planning with stochastic resource profiles: An application to human-robot cohabitation. *International Conference on Automated Planning and Scheduling (ICAPS)*.

Karpas, E.; Levine, S. J.; Yu, P.; and Williams, B. C. 2015. Robust execution of plans for human-robot teams. *International Conference on Automated Planning and Scheduling (ICAPS)* 342–346.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. *International Conference on Automated Planning and Scheduling (ICAPS)*.

Levine, S. J., and Williams, B. C. 2014. Concurrent plan recognition and execution for human-robot teams. *International Conference on Automated Planning and Scheduling (ICAPS)*.

Ramirez, M., and Geffner, H. 2009. Plan recognition as planning. *International Joint Conference on Artificial Intelligence (IJCAI)* 1778–1783.

Shah, J.; Wiken, J.; Williams, B.; and Breazeal, C. 2011. Improved human-robot team performance using chaski, a human-inspired plan execution system. *Proceedings of the 6th international conference on Human-robot interaction* 29–36.

Stubbs, K.; Wettergreen, D.; and Hinds, P. H. 2007. Autonomy and common ground in human-robot interaction: A field study. *Intelligent Systems, IEEE* 22(2):42–50.

Talamadupula, K.; Briggs, G.; Chakraborti, T.; Scheutz, M.; and Kambhampati, S. 2014. Coordination in human-robot teams using mental modeling and plan recognition. *Intelligent Robots and Systems (IROS)* 2957–2962.

Unhelkar, V. V., and Shah, J. A. 2016. Contact: Deciding to communicate during time-critical collaborative tasks in unknown, deterministic domains. *Association for the Advancement of Artificial Intelligence (AAAI)*.

Zhang, Y.; Zhuo, H. H.; and Kambhampati, S. 2015. Plan explainability and predictability for cobots. *arXiv preprint arXiv:1511.08158*.